

4

DTIC FILE COPY

CRITICAL PROBLEMS IN VERY LARGE SCALE COMPUTER SYSTEMS

AD-A215 190

Semiannual Technical Report for the Period

April 1, 1989 to September 30, 1989

Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Anant Agarwal	(617) 253-1448
William J. Dally	(617) 253-6043
Srinivas Devadas	(617) 253-0454
Thomas F. Knight, Jr.	(617) 253-7807
F. Thomson Leighton	(617) 253-3662
Charles E. Leiserson	(617) 253-5833
Paul Penfield, Jr.	(617) 253-2506
Jacob K. White	(617) 253-2543
John L. Wyatt, Jr.	(617) 253-6718

DTIC
ELECTE
DEC 04 1989
S E D

APPROVED FOR
PUBLIC DISTRIBUTION

Contents

1	Research Overview	3
2	Circuits	3
3	Processing Elements	4
4	Communications Topology and Routing Algorithms	5
5	Systems Software	6
6	Algorithms	8
7	Applications	12
8	Publications List	13
8.1	Journal and Conference Publications	13
8.2	Internal Memoranda	16
8.3	Talks without Proceedings	17

Selected Publications (starting at page 20)

W. J. Dally, "Express Cubes: Improving the Performance of the k -ary n -cube Interconnection Networks," submitted to *IEEE Transactions for Computers*. Also MIT VLSI Memo No. 89-564, October 1989.

S. Kipnis, "Priority Arbitration with Busses," MIT, September, 1989.

T. Knight, "Technologies for Low Latency Interconnection Networks," *First Annual ACM Symposium on Parallel Algorithms and Architectures*, Santa Fe, New Mexico, June 17-19, 1989.

Accession For	
NTIS GPA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	



1 Research Overview

The research vehicle for this contract is the largest possible computer that could be conceived for the mid to late 1990's. The technical challenges of such a machine serve as our guiding stimulus for the research carried out and reported here.

We imagine this machine to occupy a 14-story building, to cost upward of \$1 billion, and to be so colossal that the nation could only afford one or two of them. The available chip technology and machine size are consistent with 10^{15} FLOPS and 10^{15} bytes of memory. The machine will be used to solve large-scale scientific problems having both military and civilian applications.

This investigation addresses the hardware technology, software techniques, algorithms, communications, processing elements, and applications. The study will determine the plausibility (not feasibility) of the machine. Progress in these various areas are highlighted in the forthcoming sections.

2 Circuits

Sandy Wells and Tom Knight have designed and tested MSI prototypes of a new class of analog computing devices, based on switched capacitor constraint boxes. The core of these devices is a two-port consisting of a capacitor rapidly switched between the ports. Labelling the terminal voltages a, b, c, d , this attempts to enforce a constraint $a - b = c - d$. This is a reciprocal constraint, allowing propagation of information in either direction. We have shown that, using this basic constraint box, we can solve linear systems (to arbitrary accuracy using mixed analog/digital techniques), solve over-constrained systems with the pseudo-inverse, and solve linear programming problems. The small size, simplicity, and ease of understanding, argue that this device may be an important circuit element in next-generation hybrid computing.

Srinivas Devadas and his students have been focusing on the optimization of combinational and sequential circuits specified at the register-transfer or logic levels with area, using performance and testability of the synthesized circuit as design parameters. Work is also being done in the area of test generation for VLSI circuits.

Techniques have been proposed in the past for various types of finite state machine (FSM) decomposition that use the number of states or edges in the decomposed circuits as the cost function to be optimized. These measures are not reflective of the true logic complexity of the decomposed circuits. These methods have been mainly heuristic in nature and offer limited guarantees as to the quality of the decomposition. In this work [32], following up on our exact state assignment algorithm developed earlier [31], we have developed optimum and heuristic algorithms for the general decomposition of FSMs such that the sum total of the number of product terms in the one-hot coded and logic minimized submachines is minimum or minimal. This cost function is much more reflective of the area of an optimally state-assigned and minimized submachine than the number of states/edges in the submachine.

We are continuing to investigate the impact of logic synthesis on the testability of sequential circuits that can be modeled as finite state machines [33] [34] [37] [30]. The new approach of [34] and [37] is to use synthesis to ensure the complete testability of a sequential circuit by ensuring that each invalid state has an unperturbable distinguishing sequence. To accomplish this we have developed a Boolean minimization procedure of prime implicant generation and constrained covering based on the Quine-McCluskey algorithm that ensures that no single fault can both produce an invalid state and corrupt the distinguishing sequence by which that invalid state can be identified. On completion, it guarantees a prime and irredundant, fully testable Moore or Mealy finite state machine. Given a two-level circuit with these properties we then define constrained algebraic factorization techniques that retain the invariant that no single fault can both produce an invalid state and corrupt the distinguishing sequence by which that invalid state is detected. We have used the notion of fault-effect disjointness to explore the landscape between various synthesis approaches and have demonstrated a spectrum of methods [37] that place relatively more-or-less emphasis on either logic optimization or constrained synthesis. Techniques used in this exploration include fault simulation, Boolean covering, algebraic factorization and state assignment.

We have explored the relationships between redundant logic and don't care conditions in sequential circuits [30]. Stuck-at faults in a sequential circuit may be testable in the combinational sense, but may be redundant because they do not alter the terminal behavior of a non-scan sequential machine. These sequential redundancies result in a faulty State Transition Graph (STG) that is equivalent to the STG of

the true machine. We have precisely classified redundant faults in sequential circuits composed of single or interacting finite state machines. For each of the different classes of redundancies, we define don't care sets which if optimally exploited will result in the implicit elimination of any such redundancies in a given circuit.

We have also addressed the problem of generating test sequences for stuck-at faults in non-scan synchronous sequential circuits [38]. A novel test procedure that exploits both the structure of the combinational logic in the circuit as well as the sequential behavior of the circuit has been developed. In contrast to previous approaches, we decompose the problem of sequential test generation into three subproblems of combinational test generation, fault-free state justification and fault-free state differentiation. Initially, prior to test generation, separate sum-of-product representations of the complete or partial ON-sets and OFF-sets of each of the flip-flop inputs and primary outputs of the sequential circuit, are extracted using the PODEM algorithm. Fast algorithms for state justification and state differentiation can be based on this representation. These algorithms perform repeated cube intersections in an effort to find a justification sequence for a state or a distinguishing sequence for a pair of states.

We have also addressed the problem of generating tests for delay faults in non-scan synchronous sequential circuits [36]. Delay test generation for sequential circuits is a considerably more difficult problem than delay testing of combinational circuits and has received much less attention. We have developed a method for generating test sequences to detect delay faults in sequential circuits using a stuck-at fault sequential test generator. The method is complete in that it will generate a delay test sequence for a targeted fault given sufficient CPU time, if such a sequence exists. We term faults for which no delay test sequence exists, under our test methodology, sequentially delay redundant. We have also developed means of eliminating sequential delay redundancies in logic circuits.

Finally, we have done some preliminary work in an attempt to gain insight into the nature of NP-complete problems. In [35], we have transformed various NP-complete problems in layout, namely two and multi-layer dogleg channel routing, two-way partitioning, one-dimensional and two-dimensional placement into Boolean satisfiability problems. The transformations are efficient in that the number of inputs to the Boolean function for which we have to find a satisfying assignment, grows only linearly or quasi-linearly with the layout problem size. We have applied sophisticated test generation and logic verification strategies that can be used to check for Boolean function satisfiability to these layout problems. It appears that this approach to layout optimization offers an elegant means of representing and searching the entire space of feasible solutions in an attempt to optimize a complex cost function with associated constraints.

3 Processing Elements

The processors of a multicomputer require the ability to switch tasks rapidly to hide transmission latency without sacrificing single-thread performance. Peter Nuth and Bill Dally are working on an architecture for a named state processor that achieves this goal by explicitly binding names to all processor registers and interleaving tasks on a microcycle basis. This mechanism combines the advantages of multi-threading and multiple register sets for implementing fast context switches and procedure calls. It also provides a general synchronization mechanism.

During the past year, we have defined the named state processor architecture and its interface to a multicomputer network. We are currently studying instruction scheduling policies (deciding which processes instructions get advanced when) and context cache management policies (deciding which processes state remains in active storage). A simulator for the processor is under construction. This work is being performed by Peter Nuth as his MIT Ph.D. thesis.

Most multicomputers are specialized to execute a single model of computation (e.g., dataflow, actors or shared memory). Scott Wills and Bill Dally have identified a set of primitive mechanisms for communication, synchronization and naming that are required for all of these models of computation. We are currently evaluating these mechanisms in terms of their implementation cost and their suitability for supporting popular models of parallel computation [51] [55].

During the reporting period, we have defined a parallel machine interface that incorporates a consistent set of these mechanisms. A parallel interface simulator, PiSIM, has been constructed to facilitate experiments with the interface. Using PiSIM, dataflow and shared memory models of computation have been

implemented on the parallel machine interface. We are presently evaluating the cost and performance of these implementations.

Anant Agarwal has investigated the use of rapid-context switching VLSI RISC processors as the computing nodes in a large parallel machine. Rapid context switching allows overlapping communication and synchronization delays with computation by quickly scheduling a new process on the processor. The design of such a processor is complete. The processor, APRIL, switches between threads on either memory accesses to remote nodes, or during an unsuccessful access of a synchronization object. APRIL has tag support for Futures, and synchronization support in the form of full-empty bits associated with each memory word. APRIL also has several basic instructions to allow experimentation with a variety of shared memory programming models. These special operations include cache flushes, fences, block transfers, and user definable choice of spin-waiting versus blocking. An instruction-level simulator for APRIL has been written. A Mul-T compiler for this processor has been written and generates code that runs on the simulator. A scheduler that exploits the multithreaded nature of the processor and other run-time system software has also been written and runs on the simulator. An implementation design consisting of very minor modifications to the SPARC processor is almost complete. Because floating point operations are usually supported through the use of coprocessors in most modern day VLSI RISC microprocessors, we are investigating methods of multithreading a coprocessor. A performance evaluation of the system effects of multithreaded processors has also been completed [59]. The analytical evaluation considered the context switching overhead, and the increased cache and network contention. We showed that for most system configurations, while providing for network, cache and overhead effects, between two and four contexts were sufficient to provide close to 90 effects.

We are designing a scalable cache and memory system. A detailed protocol design for a scalable cache coherence scheme is complete and has been implemented in a simulator. A cache controller design is in progress. A VLSI implementation of the same is envisaged in the near future. The architectural and VLSI circuit design of a fast and low-storage-overhead translation scheme for processor addresses is in progress. Simulations of various cache coherence schemes such as limited directories, singly and doubly linked lists and write-through shared, are in progress. Our simulations use traces from numeric FORTRAN codes, graph algorithms written in Mul-T, and CAD applications written in C. (Our FORTRAN traces were obtained through a joint effort with IBM T. J. Watson Research Center. The Mul-T traces were obtained through a compiler-aided tracing package we wrote called T-Mul-T. We have made these traces available to other researchers also. The CAD traces are from Stanford). Initial results indicate that the performance of singly linked lists is comparable to doubly linked lists without the extra hardware overhead and complexity. Limited directories are shown to perform comparably if software support for widely-shared read-only objects and synchronization structures is provided. We wrote a novel post-mortem scheduler that can take a single-processor execution of a parallel program and, simulating the effect of various synchronization implementations such as adaptive backoff [5] or software barrier trees, produce cache statistics for the various synchronization implementations [39].

4 Communications Topology and Routing Algorithms

Bill Dally and his students are experimenting with a new flow control strategy based virtual channels. Our initial results show that this strategy can boost network throughput to 90% capacity without adaptive routing by decoupling resource constraints. Current flow control methods are limited to 30% to 50% capacity because many channels remain idle due to resource allocation coupling. This throughput limit is not due to load imbalance, which can only be addressed by adaptive routing.

The virtual channel flow control method divides a channel's flit buffers into many shallow 'lanes', rather than a single deep FIFO. The buffering is short and wide rather than long and fat. The organization decouples flit buffer resource allocation for each channel. This allows active messages to pass blocked messages that are waiting on an unrelated resource much in the way that a two lane street permits cars travelling straight ahead to pass a car that is waiting to make a left turn.

We have built a simulator of direct and indirect networks that use virtual channel flow control and have measured their performance under different loads and traffic patterns. The initial results suggest that a moderate number of virtual channels (4-8) gives a throughput that is very close to network capacity.

The remaining degradation is largely due to load imbalance and adaptive routing will be required to reach 100% capacity.

Express cubes are k-ary n-cube interconnection networks augmented by *express channels* that provide a short path for non-local messages. An express cube combines the logarithmic diameter of an indirect network with the wire-efficiency and ability to exploit locality of a direct network. The insertion of express channels reduces the network diameter and thus the distance component of network latency. Wire length is increased allowing networks to operate with latencies that approach the physical speed-of-light limitation rather than being limited by node delays. Express channels increase wire bisection in a manner that allows the bisection to be controlled independent of the choice of radix, dimension, and channel width. By increasing wire bisection to saturate the available wiring media, throughput can be substantially increased. With an express cube both latency and throughput are wire-limited and within a small factor of the physical limit on performance. Express channels may be inserted into existing interconnection networks using *interchanges*. No changes to the local communication controllers are required.

Tom Knight and his students are continuing implementation work on the Transit communication switch. We have released to manufacturing the design for the button board connector, and for the PC board component carrier. The carrier cooling technology has evolved somewhat since our last report as a result of detailed heat flow calculations. Our current approach involves flowing coolant through a microchannel heatsink bonded directly to the rear surface of the die, similar to the approach used by Tuckerman at Stanford, but at a more macroscopic level.

Die design continues, with the gate-level description and stable test-patterns, and with initial sizing and layout work under way. Initial RSIM estimates by Henry Minsky of timing (now at 17ns) indicate that substantial additional effort will be required to achieve our target of a 10ns clock rate, but we remain cautiously optimistic.

Recent design changes in the chip specification, adding a per-input-port "swallow" signal, allow the use of this design in combination with some as-yet missing packaging technology to construct much larger switching arrays based on Leiserson's fat-tree topology. Andre DeHon is actively pursuing the topological, packaging, and electrical requirements of this expansion.

Alex Ishii is incorporating recent shifts from voltage control of the pad output impedance to a scheme utilizing digitally controlled D/A networks for implementing the controlled impedance pullup and pulldown devices.

We have located commercial suppliers for closed loop Fluorinert cooling systems, and plan to purchase this component when it appears to be the pacing item in the design. High efficiency low voltage power supplies remain a difficult issue, but interim low-efficiency designs will allow us to test the remainder of the system, while determining more efficient systems.

Network design for large-scale machines was investigated by Anant Agarwal and his students. We showed that when switch delay was included in the analysis of direct interconnection networks, the optimal network implemented in two physical dimensions in terms of the latency, was three dimensional. This is in contrast to previous findings that showed that a two dimensional network was optimal. The chief reason for the difference is that node delays can make the wire delays have a relatively smaller impact on overall latency. A detailed performance model for circuit-switched interconnection networks was developed [60]. Simulators for circuit-switched and packet-switched indirect networks are operational, and we now also have a packet-switched direct network simulator.

5 Systems Software

Andrew Chien and Bill Dally are developing data abstraction tools that support the development of programs for large scale multicomputers. A language, concurrent aggregates, has been defined that facilitates the specification of aggregates of cooperating objects. Concurrent aggregates permit the relationships between objects to be defined textually rather than requiring that the objects connect up a pointer structure at run-time as is typically done. Common structures (e.g., combining trees) can be defined once and reused as required. The language also permits nesting of object aggregates and specialization of objects within the aggregate. This work is being performed by Andrew Chien for his MIT Ph.D. thesis.

During the reporting period, the concurrent aggregates (CA) language has been defined. A compiler

that translates CA programs to C++ has been written. The output of this compiler is linked with a run-time written in C++ that simulates parallel machine execution. A number of programs have been written in CA to evaluate the language. A study of the efficiency of the language and its implementation is currently underway.

Bill Dally and Lucien Van Elsen have developed a technique, micro-optimization, for reducing the operation count and time required to perform numerical calculations. The method involves first breaking floating point operations into their constituent integer micro-operations, then optimizing and scheduling the resulting integer code. The method has been tested using a prototype expression compiler [54]. We are now looking at extending the method to permit a compiler to perform automatic scaling of numbers. Where it is possible, this optimization would convert floating point expressions into integer expressions.

John Keen and Bill Dally have been investigating several problems involved in constructing highly concurrent database systems on concurrent computers augmented by large disk arrays. The goal is to develop systems technology that will permit database systems based on concurrent computers to handle 10^9 transactions per second. To date we have concentrated on parallel algorithms for logging, recovery, and consistency control. The parallel logging and recovery algorithms make use of parallel logs that represent a partial order of actions and the use of log processors to compress the logs on a regular basis. We are investigating consistency control algorithms that use reservations to achieve a higher degree of concurrency than is possible using locks.

Anant Agarwal has continued explorations of methods of programming a large-scale parallel computer such as the ARC. These investigations take two forms. First, we are looking at methods of partitioning and scheduling parallel programs to minimize communications. Numerical algorithms that can exploit locality are being investigated. Tradeoffs in the use of block techniques for linear algebraic codes are being studied. We currently have several parallel address traces of several runs of parallel blocking methods and we are studying their impact on cache and network performance. Scheduling methods that exploit both locality and the communication latency hiding, provided by a rapid context-switching processor, are being investigated. Our experimental scheduler runs on our simulation system. Our second thrust is towards enhancing our parallel programming language to allow (1) the convenient specification of data parallelism using structures similar to the dataflow I-structures, and (2) allow experimentation with data placement and relocation, function and data shipping, and different programming models including weaker shared memory models with block transfer capabilities. Our current status is that the language primitives have been defined as extensions to Mul-T and their implementation in the compiler and simulator are in progress. The APRIL compiler and linker and the lazy future kernel have been implemented. Extensions for garbage collection and efficient floating-point support are being developed. The T language has also been sorted to the Sparc and the Decstation (Pmax).

To gain more experience with programming large-scale parallel machines we are also writing several parallel applications. Our major effort has been spent on Speech. This application comprises the viterbi search portion of a connected speech recognition system being implemented by the Speech and Spoken Language Systems Group at MIT. We have also written particle-in-cell in Mul-T. Several other parallel applications that we have written include logic simulation, and permute. The Simple application is also partially written in Mul-T.

Several performance evaluation tools and methods have been developed. Our T-Mul-T multiprocessor address tracer is operational. We developed a technique for trace compaction that exploits the spatial locality of memory referencing in multiprocessors [61]. A novel model for multithreaded processors has also been derived. A processor locality-based multiprocessor cache interference model has been developed [58].

System studies putting all the above pieces together are also in progress. A detailed multiprocessor simulator has been implemented and is functional. The simulator is comprised of the APRIL processor simulator, the cache and memory system, and the interconnection network. Parallel applications written in Mul-T are compiled to APRIL code and can be executed on the multiprocessor simulator. We have successfully run our large speech application on 16 processors, each with a multithreaded degree of four. If needed the FORTRAN post-mortem scheduler or T-Mul-T tracer can replace the APRIL processor front end.

6 Algorithms

In the area of algorithms, three students—Ron Greenberg, Bruce Maggs, and Cindy Phillips—finished their Ph.D. theses under the direction of Charles Leiserson.

Ronald Greenberg has completed his Ph.D. thesis entitled "Efficient Interconnection Schemes for VLSI and Parallel Computation." The thesis is primarily concerned with the design of efficient interconnection networks for general-purpose parallel computers and the more specialized problem of multilayer channel routing for VLSI chips. In addition, it provides lower bounds on the area required for VLSI implementations of finite-state machines.

The first part of Greenberg's thesis shows why networks based on Leiserson's fat-tree architecture are nearly as good as any network built in a comparable amount of physical space. Such networks can simulate any other network of the same area with slowdown which is a small polylogarithmic function of the area. These "universal" networks can be constructed in area linear in the number of processors, so that there is no need to restrict the density of processors in competing networks. Also it is possible to compare networks that are of different size or are built from processors of different sizes (as determined by the amount of attached memory). In addition, many of the results given do not require the usual assumption of unit wire delay. Also, it is possible to simulate competing networks even if the processors are not globally synchronized into separate phases of internal computation and interprocessor communication. Finally, the results apply not only in two dimensions, but also in three dimensions by way of a simple demonstration of general results on graph layout in three dimensions. This part of the thesis includes joint work with Charles Leiserson of MIT.

The second part of Greenberg's thesis discusses the channel routing problem in the context that many layers of interconnect are available. It describes a system, MulCh, for multilayer channel routing, which extends the Chameleon system developed at U. C. Berkeley. Like Chameleon, MulCh divides a multilayer problem into essentially independent subproblems of at most three layers, but unlike Chameleon, MulCh considers the possibility of using partitions comprised of a single layer instead of only partitions of two or three layers. Experimental results show that MulCh often performs better than Chameleon in terms of channel width, total net length, and number of vias. In addition to a description of MulCh as implemented, Greenberg's thesis discusses improved algorithms for subtasks performed by MulCh, thereby indicating potential improvements in the speed and performance of multilayer channel routing. In particular, linear time suffices to determine the minimum width required for a single-layer channel routing problem, and the density of a collection of nets can be maintained in logarithmic time per net insertion. The work on MulCh is joint with Alex Ishii of MIT and Alberto Sangiovanni-Vincentelli of U. C. Berkeley; the work on single-layer channel routing is joint with Miller Maley of Princeton U.

The last part of Greenberg's thesis shows that straightforward techniques for implementing finite-state machines are optimal in the worst case. Specifically, for any s and k , there is a deterministic finite-state machine with s states and k symbols such that any layout algorithm requires $\Omega(ks \lg s)$ area to lay out its realization. For nondeterministic machines, there is an analogous lower bound of $\Omega(ks^2)$ area. This work is joint with Mike Foster of Columbia University.

Bruce Maggs also finished his dissertation, entitled *Locality in Parallel Computation*. The thesis explores strategies for exploiting locality in three major areas of parallel computation: packet routing, parallel algorithm design, and network emulations.

The first part of Maggs's thesis deals with a novel network-independent approach to the packet-routing problem. The strategy is to partition the problem into two stages: a path-selection stage and a scheduling stage. In the first stage paths are found for the packets with small congestion, c , and dilation, d . Once the paths are fixed, both are lower bounds on the time required to deliver the packets. In the second stage we find a schedule for the movement of each packet along its path so that no two packets traverse the same edge at the same time; consequently, the total time and maximum queue size required to route all of the packets to their destinations are minimized.

Although path-selection strategies vary from network to network, Maggs shows that there is an efficient on-line scheduling algorithm for the entire class of layered networks. When applied to an N -packet problem, the algorithm produces a schedule of length $O(c + d + \log N)$, with high probability. The algorithm has many applications to routing and sorting. Among them are the first on-line algorithms for routing N -packets on an N -node shuffle-exchange graph in $O(\log N)$ steps using constant-size queues and for routing

kM^k packets on a k -dimensional array with side length M in $O(kM)$ steps using constant-size queues. The scheduling algorithm can also be used as a subroutine in sorting algorithms. It yields the first asymptotically optimal algorithms for sorting on butterfly, shuffle-exchange, and multidimensional array networks using constant-size queues. The algorithm can also be applied to the construction of area-universal networks: N -node networks with VLSI-layout area $O(N)$ that can simulate all other networks with area $O(N)$ with only $O(\log N)$ slowdown. Maggs also proves the existence of a schedule of length $O(c + d)$ for any set of packets whose paths have congestion c and dilation d (in any network) that uses constant-size queues. Unfortunately, no efficient algorithm for constructing the schedule is known.

The second part of Maggs's thesis introduces a model for parallel computation, called the *distributed random-access machine* (DRAM), in which the communication requirements of parallel algorithms can be evaluated. A DRAM is an abstraction of a parallel computer in which memory accesses are implemented by routing messages through a communication network. It explicitly models the congestion of messages across cuts of the network.

Maggs introduces the notion of a *conservative* algorithm as one whose communication requirements at each step can be bounded by the congestion of pointers of the input data structure across cuts of a DRAM. A conservative algorithm is guaranteed not to generate undo congestion in any underlying network. Maggs presents conservative algorithms for a variety of graph problems. Problems such as computing treewalk numberings, finding the separator of a tree, and evaluating all subexpressions in an expression tree can be solved in $O(\log N)$ steps for N -node trees by conservative algorithms for an exclusive-read exclusive-write DRAM. More complex problems include finding a minimum-cost spanning forest, and computing biconnected components and constructing an Eulerian cycle require $O(\log^2 N)$ steps, for graphs of size N . For concurrent-read concurrent-write DRAM's, all of these problems can be solved by $O(\log N)$ step conservative algorithms.

The final part of the thesis examines the problem of how efficiently a host network can emulate a guest network. The goal is to emulate T_G steps of an N_G -node guest network on an N_H node host network. An emulation is called *work-preserving* if the time required by the host, T_H is $O(T_G N_G / N_H)$ because then both the guest and host networks perform the same amount of total work (processor-time product), $\Theta(T_G N_G)$, to within a constant factor. A work-preserving emulation is *efficient* because it achieves optimal speedup over a sequential emulation of the guest. An emulation is *real-time* if $T_H = O(T_G)$, because then the host emulates the guest with constant delay.

Although many isolated emulation results have been proved for specific networks in the past, and measures such as dilation and congestion were known to be important, the field has lacked a model within which general results and meaningful lower bounds could be proved. Maggs provides such a model, along with techniques for proving lower bounds based on comparing the locality the networks. Some of the more interesting and diverse results in this part of the thesis include a proof that a linear array can emulate a (much larger) butterfly in a work-preserving fashion, but that a butterfly cannot emulate an expander (of any size) in a work-preserving fashion; a proof that a mesh can be emulated in real time in a work-preserving fashion on a butterfly, even though any $O(1)$ -to-1 embedding of the mesh has dilation $\Omega(\log N)$; and a proof that an N -node butterfly can emulate an $N \log N$ -node shuffle-exchange graph in a work-preserving fashion, and vice-versa.

Cynthia Phillips finished her dissertation, entitled *Theoretical and Experimental Analyses of Parallel Combinatorial Algorithms*. The thesis investigates parallel algorithms for graph and matrix problems. Some of the algorithms are known, and some she has developed. She has analyzed them theoretically and experimentally. The thesis is broken into five parts.

The first major contribution of her thesis shows how n -node, e -edge graphs can be *contracted* in a manner similar to the parallel tree contraction algorithm due to Miller and Reif. She gives an $O((n + e) / \lg n)$ -processor deterministic algorithm that contracts a graph in $O(\lg^2 n)$ time in the EREW PRAM model. She also gives an $O(n / \lg n)$ -processor randomized algorithm that with high probability can contract a bounded-degree graph in $O(\lg n + \lg^2 \gamma)$ time, where γ is the maximum genus of any connected component of the graph. (The algorithm can be made to run in deterministic $O(\lg n \lg^* n + \lg^2 \gamma)$ time using known techniques.) This algorithm does not require *a priori* knowledge of the genus of the graph to be contracted. The contraction algorithm for bounded-degree graphs can be used directly to solve the problem of region labeling in vision systems, i.e., determining the connected components of bounded-degree planar graphs in $O(\lg n)$ time, thus improving the best previous bound of $O(\lg^2 n)$.

The second part describes four APL-like primitives for manipulating dense matrices and vectors and describe their implementation on the Connection Machine hypercube multiprocessor. These primitives provide a natural way of specifying parallel matrix algorithms independently of machine size or architecture and can actually enhance efficiency by facilitating automatic load balancing. The implementations are efficient in the frequently occurring case where there are fewer processors than matrix elements. In particular, if there are $m > plgp$ matrix elements, where p is the number of processors, then the implementations of some of the primitives are asymptotically optimal for a weak hypercube in that the processor-time product is no more than a constant factor higher than the running time of the best serial algorithm. Furthermore, the parallel time required is optimal to within a constant factor. Her implementation of the primitives on the Connection Machine 2 system improved the performance of a simplex program for linear programming by almost an order of magnitude over a naive implementation, from 55 Mflops to 525 Mflops.

The third portion of her thesis investigates *dimension-exchange load balancing* which is a generalization of one of the techniques used in the hypercube implementation of the vector-matrix primitives. She shows that when tasks are considered indivisible, after one pass of dimension-exchange load balancing, in the worst case, some processor will have $\Theta(\lg n)$ tasks over the average. She also shows that there is an initial distribution of tasks for which this load-balancing strategy requires an average of $\Theta(\lg n)$ messages for each unit reduction in the global maximum number of tasks.

The fourth part of Phillips's thesis reports on preliminary experimental investigations which indicate that massively parallel computers like the Connection Machine (CM) appear to be well suited for both sparse and dense implementations of dual relaxation algorithms for network optimization. (Her parallel implementation of a nonlinear network optimization program on the Connection Machine is the fastest program to date for its class of problems.) Implementations of a dense version of a known algorithm for the assignment problem and parallel versions of known heuristics for the traveling salesman problem suffered from a "sequential tail" phenomenon. Tail-cutting heuristics with appropriate (case-sensitive) parameters improved performance markedly.

The fifth and last contribution in her thesis is the design of a VLSI chip which pseudorandomly permutes bit-serial messages by sending them through a Benes network whose switches have been pseudorandomly set. Providing a pseudorandom permuter in a simple, high-throughput chip could improve the performance of routing algorithms for multiprocessors.

Shlomo Kipnis investigated priority arbitration schemes that employ busses to arbitrate among n modules in a digital system. He focused on distributed mechanisms that employ m busses, for $\lg n \leq m \leq n$, and use asynchronous combinational arbitration logic. A widely used distributed asynchronous mechanism is the *binary arbitration* scheme, which with $m = \lg n$ busses arbitrates in $t = \lg n$ units of time. Shlomo Kipnis presented a new asynchronous scheme — *binomial arbitration* — that by using $m = \lg n + 1$ busses reduces the arbitration time to $t = \frac{1}{2} \lg n$. Extending this result, he presented the *generalized binomial arbitration* scheme that achieves a bus-time tradeoff of the form $m = \Theta(tn^{1/t})$ between the number of arbitration busses m and the arbitration time t (in units of bus-settling delay), for values of $1 \leq t \leq \lg n$ and $\lg n \leq m \leq n$. These schemes are based on a novel analysis of *data-dependent delays* and generalize the two known schemes: *linear arbitration*, which with $m = n$ busses achieves $t = 1$ time, and *binary arbitration*, which with $m = \lg n$ busses achieves $t = \lg n$ time. Most importantly, these schemes can be adopted with no changes to existing hardware and protocols; they merely involve selecting a *good* set of priority arbitration codewords. The *binomial arbitration* and the *generalized binomial arbitration* schemes are a subject of a patent application.

Bruce Maggs and Tom Leighton have been studying adaptive fault-tolerant algorithms for packet routing. They have shown that an N -input multibutterfly can sustain k faults and still route $\log N$ permutations between some set of $N - O(k)$ inputs and $N - O(k)$ outputs in $O(\log N)$ time. The multibutterfly is even more resilient to randomized faults. For example, with high probability, a specially modified twin butterfly can tolerate $N^{3/4}$ faulty internal nodes, and still route any $\log N$ permutations of N packets in $O(\log N)$ time. Thus, the multibutterfly is the first bounded-degree network known to be able to sustain large numbers of faults with only minimal degradation in performance.

In the past year, Tom Cormen has continued to write the textbook *Introduction to Algorithms* with Professors Leiserson and Rivest. The book will be published in early 1990.

Marios Papaefthymiou continued his research on synchronous circuit optimization under the supervision

of Prof. Leiserson. His work focused on investigating the underlying structure of the retiming operation. The result of this effort was a concise closed-semiring description of retiming for unit-delay circuits. This compact description suggests a promising point of view for looking at retiming. Marios Papaefthymiou is currently trying to design efficient algorithms for optimum retiming, by exploiting the group structure that he revealed.

During the past six months, James K. Park has been collaborating with Alok Aggarwal and Dina Kravets on a number of problems relating to totally monotone arrays. Such arrays arise naturally in a wide variety of fields, including computational geometry, dynamic programming, and VLSI river routing. Park's work with Aggarwal centers on the problem of finding maximum entries in totally monotone arrays and applications of efficient sequential and parallel algorithms for this problem. Park's work with Kravets investigates the problems of selection and sorting in the context of totally monotone arrays and applications of efficient algorithms for these problems.

Alexander Ishii has been generalizing his VLSI timing analysis algorithms. A key concern has been the need to accurately handle the "undefined" values that electrical signals must take on when they make a transition between valid logic levels. In addition, he has attempted to make the algorithms easily adaptable to different assumptions about the circuit being analyzed.

Prof. Leighton is continuing his research on networks and algorithms for parallel computation. Recently he has focussed on the following specific problems: the development of fast packet routing algorithms for commonly used fixed-connection networks, the development of algorithms to reconfigure networks such as the hypercube around faults, the development of dynamic on-line algorithms for embedding computational structures such as trees in networks, such as the hypercube, in a way that balances computational load and that minimizes the induced communication load on the network, the development of algorithms for emulating one kind of network on another in a way that preserves the total amount of work (processors \times time) that is done, and the development of a new network architecture for routing that can tolerate large numbers of faults without a substantial degradation in performance. The particular advances that have been made in each of these areas is briefly summarized in what follows.

In the area of packet routing, Prof. Leighton and his coauthors have discovered the first store-and-forward routing algorithm which can route n^2 packets in $2n - 2$ steps on an $n \times n$ array with constant size queues at each node. The details of these and related results can be found in [16]. They have also discovered new and more efficient routing algorithms for the multibutterfly. These algorithms are the first that are highly tolerant of worst case faults. Also in the area of fault-tolerance, Prof. Leighton and his coauthors have shown that a hypercube can tolerate a very large number (a constant fraction) of randomly located faults without incurring more than a constant factor loss in performance, no matter how large the hypercube is. They have also discovered simple algorithms for routing around faults in the hypercube that are guaranteed to perform nearly as well as the best routing algorithms when no faults are present. The details of this work are described in [12].

In the area of network embeddings and scheduling, Prof. Leighton and his coauthors have discovered optimal algorithms for embedding dynamically growing and shrinking trees in a hypercube so that the processing load on the nodes of the hypercube is balanced, and so that all communication links are local. This work has application to the problem of locally scheduling the work assigned to the processors of a hypercube in a dynamic fashion (i.e., as one computation spawns another, the algorithm determines the processor that will handle the new task). They have also discovered optimal algorithms for mapping code written for one architecture onto a different architecture in a way that minimizes the total amount of work required by the simulating machine. These results are described in [7,25].

The past year was also a good one for Prof. Leighton's students. Bruce Maggs, Satish Rao, Richard Koch, and Mark Newman all obtained their Ph.D.s this year. Together with Prof. Leighton, they made lots of solid progress on packet routing algorithms, fault tolerance in networks, and on graph embedding problems. At this point they are getting close to asymptotically optimal results that also appear to work well in reality. In fact, the highlight of the coming year will be to help design and lay out a multibutterfly network for Tom Knight's new machine. With a little luck, theory will be able to play an important role in the development of a state of the art machine. Prof. Leighton is also working with Bill Dally and his students to see if theory can be helpful with the routing protocols on his new machine, and he has been talking with Alan Baratz about the possibilities of implementing some of the new theory routing algorithms on the IBM GF11 so that it can become a general purpose routing machine.

Another highlight of the last six months was the new ACM Symposium on Parallel Algorithms and Architectures that Prof. Leighton helped to organize. The first meeting was in Santa Fe in mid-June, and the meeting was very successful. Papers that were presented ranged from theory to practice and the meeting provided a good forum for interaction between people who think about parallel machines, those who build them, and those who use them.

7 Applications

Over the past six months, efforts in developing numerical algorithms for problems related to the design of an ARC, as well as those that can effectively exploit the ARC's capability, have continued under the direction of Jacob White. Interesting new algorithms have been unearthed in the areas of parallel circuit simulation and monte carlo device simulation. In addition, preliminary experiments with recently developed algorithms in capacitance extraction and classical semiconductor device simulation have been completed with very encouraging results.

In the area of circuit simulation, we have completed the development of SIMLAB [69,70], a fast, general purpose circuit simulation program intended for use in circuit simulation research. The program is presently being used for our course in numerical simulation as well as forming the basis for three ongoing research projects. SIMLAB is being used to study multiple timepoint methods for increasing the parallelism in circuit simulation so as to effectively exploit a massively parallel processor on reasonable sized problems. In addition, SIMLAB is being used to study multigrid variations for efficient simulation of the analog arrays, like those used in early vision.

SIMLAB has also been used to study the behavior of the switched linear resistive and nonlinear resistive networks used for image smoothing and segmentation algorithms (under the supervision of Prof. J. Wyatt). Arc-length style continuation methods were added to SIMLAB so that comparison studies of several continuation methods can be gracefully implemented in analog VLSI.

Also in the area of circuit simulation, we have undertaken a study of Exponential-Fitting numerical integration algorithms. We have been able to prove several strong results indicating that the performance of recently published exponential-fitting algorithms are, in the limit of large timesteps, identical to other well-known techniques. Detailed experiments indicate exponential-fitting offers little advantage. We have also examined several modifications which seem to improve the accuracy of the exponential-fitting algorithm, but it is unlikely to produce results that are competitive with more standard techniques.

In the area of classical device simulation, we have completed preliminary experiments using waveform relaxation to perform transient two-dimensional simulation of MOS devices. Experiments demonstrate that WR converges in a uniform manner, and that there is typically some multirate behavior in a device that the WR algorithm can exploit. Speed and accuracy comparisons between standard direct methods, red/black Gauss-Seidel WR, and red/black overrelaxed WR indicate that for the experiments examined, calculated terminal currents match well between the methods, and that overrelaxed WR was between 2 and 5 times faster than direct methods. A recently implemented modification based on a waveform-Newton algorithm increased this to a factor of from 5 to 11 [9,10].

Our other project in classical device simulation is in developing efficient and robust numerical algorithms for a two-dimensional semiconductor device simulator that includes both momentum and energy balance equations. Tracking the electron energies allows for a more accurate characterization of both hot electron effects and substrate currents. The program developed uses a full Newton method to compute potentials, electron concentrations, and electron temperatures on a grid that describes the device. Initial simulation results on a MOSFET were close to what was expected theoretically and what had been published in the literature by other researchers. Because of the reliability of the algorithms used in this program, we expect to be able to examine the effects of a wider range of physical models for mobility and impact ionization.

Simulation of small geometry devices by particle simulation or Monte-Carlo techniques is becoming increasingly popular, even though the method is computationally much more expensive than numerically solving the standard or modified drift-diffusion equations. We are presently investigating alternative numerical techniques to see if it is possible to make Monte-Carlo simulation less computationally expensive and more parallelizable. In particular, we are investigating the interaction between the particle motions

and the changes in the electric fields.

Three dimensional capacitance and inductance extraction have recently become important because the dense packing of processors and the memory required for high performance parallel computers require three dimensional interconnection. To insure an interconnect design will be capable of achieving desired performance, coupling capacitance and inductance must be examined. Over the past year we developed a capacitance extraction algorithm for arbitrary geometries of ideal conductors in a uniform dielectric. The algorithm reduces the calculation complexity from order n^3 , for the standard algorithm, to order n , where n is the number of tiles the into which conductor surfaces are discretized. The algorithm uses a combination of an iterative technique and a multipole expansion algorithm. The initial stages in the implementation and testing of a fast multipole accelerated conjugate gradient algorithm for extraction of capacitances from complex three dimensional geometries are complete, and the method provides nearly an order of magnitude speed improvement of the standard approach with as few as eight conductors [8].

8 Publications List

8.1 Journal and Conference Publications

1. R. R. Koch, *An Analysis of the Performance of Interconnection Networks for Multiprocessor Systems*, Ph.D. Thesis, Department of Mathematics, MIT, May 1989. Also, MIT VLSI Memo No. 89-561, September 1989.
2. M. Newman, *Randomness and Robustness in Hypercube Computation*, Ph.D. Thesis, Department of Mathematics, MIT, September 1989.
3. S. Rao, *Finding Small Balanced Edge Cuts: Theory and Applications*, Ph.D. Thesis, Department of Electrical Engineering and Computer Science, MIT, September 1989.
4. S. Owicki and A. Agarwal, "Evaluating the Performance of Software Cache Coherence," *Proceedings of ACM Architectural Support for Programming Languages and Operating Systems-III*, Boston, Massachusetts, April 1989. Also MIT VLSI Memo No. 88-478, October 1988.
5. A. Agarwal and M. Cherian, "Adaptive Backoff Synchronization Techniques," *Proceedings of the 16th Annual Symposium on Theory of Computing, ACM, SIGARCH*, June 1989. Also, MIT VLSI Memo No. 89-547, July 1989.
6. K. Kundert, J. White, and A. Sangiovanni-Vincentelli, "A Mixed Frequency-Time Approach for Distortion Analysis of Switched Capacitor Filters," *IEEE Journal of Solid State Circuits*, April 1989. Also MIT VLSI Memo No. 88-480, October 1988.
7. R. Koch, T. Leighton, B. Maggs, S. Rao, and A. Rosenberg, "Work-preserving Emulations of Fixed-connection Networks," *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, May 1989. Also MIT VLSI Memo No. 89-531, May 1989.
8. K. Nabors and J. White, "A Fast Multipole Algorithm for Capacitance Extraction of Complex 3-D Geometries," *Proceedings, Custom Integrated Circuits Conference*, San Diego, California, May 16-18, 1989. Also, MIT VLSI Memo No. 89-508, February 1989.
9. J. White and M. Reichelt, "Techniques for Switching Power Converter Simulation," *NASECODE Conference*, Dublin, Ireland, July 1989.
10. M. Reichelt, J. White, and J. Allen, "Waveform Relaxation for Two-Dimensional Device Transient Simulation," to appear in *Proceedings of the International Conference on Computer-Aided Design*, Santa Clara, California, November 1989.
11. M. Crow, J. White, and M. Ilic, "Convergence Properties of the Waveform Relaxation Method as Applied to Electric Power Systems," *Proceedings, International Symposium on Circuits and Systems*, Portland, Oregon, May 8-11, 1989.

12. J. Hastad, T. Leighton, M. Newman, "Fast Computation using Faulty Hypercubes," *Proceedings, ACM Symposium on Theory of Computing*, May 1989. Also MIT VLSI Memo No. 89-546, May 1989.
13. T. Knight, "Technologies for Low Latency Interconnection Networks," *First Annual ACM Symposium on Parallel Algorithms and Architectures*, Santa Fe, New Mexico, June 17-19, 1989.
14. C. Phillips, "Parallel Graph Contraction," *First Annual ACM Symposium on Parallel Algorithms and Architectures*, June 1989. Also MIT VLSI Memo No. 89-535, May 1989.
15. A. Agrawal, G. Blleloch, R. Krawitz, and C. Phillips, "Four Vector-matrix Primitives," *First Annual ACM Symposium on Parallel Algorithms and Architectures*, June 1989. Also MIT VLSI Memo No. 89-536, May 1989.
16. T. Leighton, F. Makedon, and I. Tollis, "A $2n - 2$ Step Algorithm for Routing in an $n \times n$ Array with Constant-size Queues," *First Annual ACM Symposium on Parallel Algorithms and Architectures*, June 1989.
17. T. Bu, C. Heigham, C. Jones, and T. Leighton, "Improving the Performance of the Kernighan-Lin and Simulated Annealing Graph Bisection Algorithms," *Proceedings of the 26th Design Automation Conference*, Las Vegas, Nevada, June 1989. Also MIT VLSI Memo No. 89-549, June 1989.
18. S. Devadas, "Approaches to Multi-level Sequential Logic Synthesis," *Proceedings of the 26th Design Automation Conference*, Las Vegas, Nevada, June 1989. Also MIT VLSI Memo No. 89-509, February, 1989.
19. S. Devadas, "General Decomposition of Sequential Machines: Relationships to State Assignment," *Proceedings of the 26th Design Automation Conference*, Las Vegas, Nevada, June 1989. Also MIT VLSI Memo No. 89-510, March 1989.
20. W. J. Dally, "The J-Machine: System Support for Actors," *Concurrent Object Programming for Knowledge Processing: An Actor Perspective*, C. Hewitt and G. Agha, editors, MIT Press, 1989. Also MIT VLSI Memo No. 88-491, December 1988.
21. W. Horwat, A. A. Chien, and W. J. Dally, "Experience with CST: Programming and Implementation," *Proceedings of the ACM SIGPLAN '89 Conference on Programming Language Design and Implementation*, 1989. Also MIT VLSI Memo No. 89-530, May 1989.
22. S. Bhatt, F. Chung, T. Leighton, and A. Rosenberg, "Universal Graphs for Bounded-degree Trees and Planar Graphs," *SIAM Journal of Discrete Mathematics*, Vol. 2, No. 2, May 1989, pp. 145-155.
23. T. Leighton and P. Shor, "Tight Bounds for Minimax Grid Matching, with Applications to the Average Case Analysis of Algorithms," *Combinatorica*, Vol. 9, 1989, pp. 179-205.
24. T. Leighton, D. Pathria, and F. Makedon, "Efficient Reconfiguration of WSI Arrays," to appear in *Proceedings of the first ACM/IEEE International Conference on System Integration*.
25. T. Leighton, M. Newman, A. Ranade, and E. Schwabe, "Dynamic Tree Embeddings in Butterflies and Hypercubes," *Proceedings of the First ACM Symposium on Parallel Algorithms and Architectures*, June 1989, pp. 224-234.
26. D. L. Standley, and J. L. Wyatt, Jr., "Stability Criterion for Lateral Inhibition and Related Networks that is Robust in the Presence of Integrated Circuit Parasitics," *IEEE Transactions on Circuits and Systems*, Vol. 36, No. 5, May 1989. Also MIT VLSI Memo No. 88-494, November 1988.
27. T. Leighton, A. Aggarwal and K. Palem, "Area-Time Optimal Circuits for Iterated Addition in VLSI," to appear in *IEEE Transactions on Computers*.

28. J. Harris, C. Koch, J. Luo, and J. L. Wyatt, Jr., "Resistive Fuses: Analog Hardware for Detecting Discontinuities in Early Vision," to appear in *Analog VLSI Implementations of Neural Systems*, C. Mead and M. Ismail, eds., Kluwer, Norwell, Massachusetts, 1989. Also MIT VLSI Memo No. 89-551, June 1989.
29. S. Devadas, H.-K. T. Ma, and A. R. Newton, "Easily Testable PLA-based Finite State Machines," *Proceedings, FTCS-19*, June 1989. Also MIT VLSI Memo No. 89-514, March 1989.
30. S. Devadas, H.-K. T. Ma, and A. R. Newton, "Redundancies and Don't Cares in Sequential Logic Synthesis," *Proceedings, International Test Conference*, Washington, D.C., August 1989. Also MIT VLSI Memo No. 89-538, May 1989.
31. S. Devadas and A. R. Newton, "Exact Algorithms for Output Encoding, State Assignment and Four-Level Boolean Minimization," *Proceedings of 23rd Hawaii Conference on System Sciences*, Hawaii, January 1990. Also MIT VLSI Memo No. 89-569, October 1989.
32. P. Ashar, S. Devadas and A. R. Newton, "Optimum and Heuristic Algorithms for Finite State Machine Decomposition and Partitioning," *Proceedings of International Conference on Computer-Aided Design*, Santa Clara, November 1989. Also MIT VLSI Memo No. 89-558, September 1989.
33. S. Devadas et. al, "Irredundant Sequential Machines Via Optimal Logic Synthesis," *Proceedings of 23rd Hawaii Conference on System Sciences*, Hawaii, January 1990. Also MIT VLSI Memo No. 89-571, October 1989.
34. S. Devadas and K. Keutzer, "Boolean Minimization and Algebraic Factorization Procedures for Fully Testable Sequential Machines," *Proceedings of International Conference on Computer-Aided Design*, Santa Clara, November 1989. Also MIT VLSI Memo No. 89-557, September 1989.
35. S. Devadas, "Optimal Layout Via Boolean Satisfiability," *Proceedings of International Conference on Computer-Aided Design*, Santa Clara, November 1989. Also MIT VLSI Memo No. 89-557, September 1989.
36. S. Devadas, "Delay Test Generation for Synchronous Sequential Circuits," *Proceedings of International Test Conference*, Washington D. C., August 1989. Also MIT VLSI Memo No. 89-539, May 1989.
37. S. Devadas and K. Keutzer, "A Unified Approach to the Synthesis of Fully Testable Sequential Machines," *Proceedings of the 23rd Hawaii Conference on System Sciences*, Hawaii, January 1990. Also MIT VLSI Memo No. 89-570, October 1989.
38. A. Ghosh, S. Devadas and A. R. Newton, "Test Generation for Highly Sequential Circuits," *Proceedings of International Conference on Computer-Aided Design*, Santa Clara, November 1989. Also MIT VLSI Memo No. 89-553, August 1989.
39. M. M. Cherian, "A Study of Backoff Barrier Synchronization," M.S. Thesis, Department of Electrical Engineering and Computer Science, MIT, May 19, 1989. Also MIT VLSI Memo No. 89-560, September 1989.
40. C. Phillips, "Theoretical and Experimental Analysis of Parallel Combinatorial Algorithms," Ph.D. Thesis, Department of Electrical Engineering and Computer Science, MIT, October, 1989.
41. R.I. Greenberg, "Efficient Interconnection Schemes for VLSI and Parallel Computation," Ph.D. Thesis, Department of Electrical Engineering and Computer Science, MIT, August, 1989.
42. B.M. Maggs, "Locality in Parallel Computation," Ph.D. Thesis, Department of Electrical Engineering and Computer Science, MIT, September, 1989.
43. D. Kravets and J. Park, "Selection and Sorting in Totally Monotone Arrays," to appear in *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms*, January 1990.

44. J. L. Wyatt, Jr., D. L. Standley, "Criteria for Robust Stability In A Class Of Lateral Inhibition Networks Coupled Through Resistive Grids," *Neural Computation*, Vol. 1, 1989, pp.58-67. Also MIT VLSI Memo No. 88-493, November 1988.
45. A. Agarwal and A. Gupta, "Temporal, Processor, and Spatial Locality in Multiprocessor Memory References," to appear in *Frontiers in Computing Systems Research*, Plenum Press, Stu Tewksbury, Ed, 1989. Also MIT VLSI Memo No. 89-519, March 1989 and MIT-LCS Technical Memo No. 397, June 1989.
46. W. J. Dally, "A Fast Translation Method for Paging on top of Segmentation," to appear in *IEEE Transactions on Computers*. Also MIT VLSI Memo No. 89-502, January 1989.
47. W. J. Dally and P. Agrawal, "A Hardware Logic Simulation System," to appear in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.
48. W. J. Dally, "Performance Analysis of k -ary n -cube Interconnection Networks," to appear in *IEEE Transactions on Computers*. Also MIT VLSI Memo No. 87-424, November 1987.
49. W. J. Dally, "Processor Design for Concurrent Computers," *Proceedings of the Canadian Conference on VLSI*, October, 1989.
50. W. J. Dally, et.al., "The J-Machine: A Fine-Grain Concurrent Computer," *Information Processing 89*, Elsevier Science Publishers, IFIP, 1989. Also MIT VLSI Memo No. 89-532, May 1989.
51. W. J. Dally and D. S. Wills, "Universal Mechanisms for Concurrency," *Lecture Notes in Computer Science*, edited by E. Odijk, M. Rem and J. C. Syre, Springer-Verlag Publishers, PARLE '89, vol. 365, June 1989, pp. 19-33.
52. P. Agrawal, R. Tutundjian and W. J. Dally, "Algorithms for Accuracy Enhancement in a Hardware Logic Simulator," *Proceedings of the 26th ACM/IEEE Design Automation Conference*, June, 1989.
53. W. Horwat, A. Chien and W. J. Dally, "Experience with CST: Programming and Implementation," *Proceedings of the ACM SIGPLAN '89 Conference on Programming Language Design and Implementation*, vol. 24, no. 7, July 1989, pp. 101-109. Also MIT VLSI Memo No. 89-530, May 1989.
54. W. J. Dally, "Micro-Optimization of Floating-Point Operations," *Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 1989, pp. 283-289. Also MIT VLSI Memo No. 88-470, August 1988.
55. W. J. Dally, "Mechanisms for Concurrent Computing," *Proceedings of the International Conference on Fifth Generation Computer Systems*, edited by ICOT, vol. 1, pp. 154-156, 1988.
56. W. Horwat, *Concurrent Smalltalk on the Message-Driven Processor*, M.S. Thesis, Department of Electrical Engineering and Computer Science, MIT, May 1989.
57. L. A. Glasser, A. C. Malamy, and C. W. Selvidge, "A Magnetic Power and Communication Interface for a CMOS Integrated Circuit," *IEEE Journal of Solid-State Circuits*, Vol 24, No. 4, August 1989, pp. 1146-1149.

8.2 Internal Memoranda

58. A. Agarwal, "A Locality-Based Multiprocessor Cache Interference Model," MIT, September, 1989. Also MIT VLSI Memo No. 89-565, October 1989.
59. A. Agarwal, "Performances Tradeoffs in Multithreaded Processors," MIT, September, 1989. Also MIT VLSI Memo No. 89-566, October 1989.
60. D. Nussbaum, I. Vuong-Adlerberg, and A. Agarwal, "Modeling Circuit Switched Multiprocessor Interconnect," MIT, September, 1989. Also MIT VLSI Memo No. 89-567, October 1989.

61. A. Agarwal and M. Huffman, "Blocking: Exploiting Spatial Locality for Trace Compaction," MIT, September, 1989.
62. B. Maggs, T. Leighton, "Expanders Might Be Practical: Fast Algorithms for Routing Around Faults on Multibutterflies," MIT, September, 1989.
63. S. Kipnis, "Priority Arbitration with Busses," MIT, September, 1989.
64. A. Aggarwal and J. Park, "Sequential Searching in Multidimensional Monotone Arrays," Submitted for publication to *Journal of Algorithms*.
65. A. Aggarwal and J. Park, "Parallel Searching in Multidimensional Monotone Arrays," Submitted for publication to *Journal of Algorithms*.
66. W. Horwat, B. Totty, and W. J. Dally, "COSMOS: An Operating System for a Fine-Grain Concurrent Computer," Submitted for publication to *IEEE Computer*.
67. W. J. Dally, "Express Cubes: Improving the Performance of the k -ary n -cube Interconnection Networks," submitted to *IEEE Transactions for Computers*. Also MIT VLSI Memo No. 89-564, October 1989.
68. Andrew Chien and William J. Dally, "Concurrent Aggregates (CA)," submitted to *Proceedings of the Second ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*.
69. A. Lumsdaine, L. Silveira, J. White, "SIMLAB Users's Manual."
70. A. Lumsdaine, L. Silveira, J. White, "SIMLAB Programmer's Manual."

8.3 Talks without Proceedings

1. T. F. Knight, Jr., "The Transit Interconnection Network," ISCA 1989 Workshop, Eilat, Israel, June 3, 1989.
2. T. F. Knight, Jr., "Solving Linear Equations with Switched Capacitor Constraints," Caltech, Pasadena, California, September 6, 1989.
3. T. F. Knight, Jr., "Solving Linear Equations with Switched Capacitor Constraints," presented at the MIT Artificial Intelligence Laboratory, Cambridge, Massachusetts, September 26, 1989.
4. A. Agarwal, "ALEWIFE: A Scalable Cache-Coherent Multiprocessor: Design Issues Solutions, and Wild Ideas," UC Berkeley, Department of Electrical Engineering and Computer Science, Berkeley, California, June 1989.
5. A. Agarwal, "ALEWIFE: A Scalable Cache-Coherent Multiprocessor: Design Issues Solutions, and Wild Ideas," Stanford University, Computer Systems Laboratory, Palo Alto, California, June 1989.
6. A. Agarwal, "ALEWIFE: A Scalable Cache-Coherent Multiprocessor: Design Issues Solutions, and Wild Ideas," Systems Research Center, Digital Equipment Corporation, Palo Alto, California, June 1989.
7. M. Cherian, "Adaptive Backoff Synchronization Techniques," International Computer Architecture Conference, Jerusalem, Israel, June 1989.
8. G. Maa, "Performance Tradeoffs in Interconnection Network Design for Large-Scale Multiprocessors," MIT VLSI Research Review, Cambridge, Massachusetts, May 22, 1989.
9. J. L. Wyatt, Jr., "The MIT Vision Chip Project," Wharton Club of Boston Annual Meeting, Boston, Massachusetts, May 1989.
10. J. L. Wyatt, Jr., "On Artificial Intelligence, Neural Networks, and the Chipmunk," The Symposium, MIT, Cambridge, Massachusetts, September 1989.

11. R. Greenberg, "MulCh: A Multi-Layer Channel Router Using One, Two, and Three Layer Partitions," MIT VLSI Research Review, MIT, Cambridge, Massachusetts, May 22, 1989.
12. R. Greenberg, "Efficient Multi-Layer Channel Routing," Georgia Institute of Technology, University of Maryland, March-April 1989.
13. R. Greenberg, "Area-Universal Networks," University of Southern California, March 1989.
14. S. Kipnis, C. Leiserson and J. Kilian, "The Organization of Permutation Architectures with Bussed Interconnections," IBM Hawthorne, August 30, 1989.
15. B. Maggs, "Work-Preserving Emulations of Fixed-Connection Networks," DARPA contractors meeting, Utah, April 3, 1989.
16. B. Maggs, "Work-Preserving Emulations of Fixed-Connection Networks," presented at the 21st Annual ACM Symposium on Theory and Computing, Seattle, Washington, May 16, 1989.
17. W. J. Dally, "Concurrent Programming," presented at Intel Corporation, Santa Clara, California, August, 1989.
18. W. J. Dally, "The J-Machine: A Fine-Grain Concurrent Computer," presented at IFIP '89, 11th World Computer Congress, August, 1989.
19. W. J. Dally, "Universal Mechanisms for Concurrency," presented at DARPA Parallel Processing Workshop, University of Maryland, College Park, Maryland, August, 1989.
20. W. J. Dally, "The J-Machine," presented at Intel Corporation, Santa Clara, California, August, 1989.
21. W. J. Dally, "Universal Mechanisms for Concurrency," presented at AT&T Bell Labs, Murray Hill, New Jersey, July, 1989.
22. W. J. Dally, "Universal Mechanisms for Concurrency," presented at PARLE '89 Conference, Eindhoven, The Netherlands, June, 1989.
23. W. J. Dally, "The Message-Driven Processor," presented at Intel Corporation, Santa Clara, California, June 1989.
24. W. J. Dally, "The J-Machine," presented at Supercomputing Research Center, Bowie, Maryland, June, 1989.
25. W. J. Dally, "Experience with CST: Programming and Implementation," presented at SIGPLAN '89 Conference, Portland, Oregon, June 1989.
26. W. J. Dally, "The J-Machine," presented at Distinguished Lecture Series in Computer Architecture, Northeastern University, Boston, Massachusetts, May, 1989.
27. W. J. Dally, "Parallel Processing: Architecture and Directions," presented at IBM T.J. Watson Research Center, Yorktown Heights, New York, May 1989.
28. W. J. Dally, "Fine-Grain Concurrent Computing," presented at IBM T.J. Watson Research Center, Yorktown Heights, New York, April 1989.
29. W. J. Dally, "The J-Machine," presented at IBM T.J. Watson Research Center, Yorktown Heights, New York, April 1989.
30. W. J. Dally, "Micro-Optimization of Floating-Point Operations," presented at ASPLOS-III, Boston, Massachusetts, April, 1989.
31. W. J. Dally, "The J-Machine," presented at Intel Corporation, Portland, Oregon, January, 1989.

32. J. White, "Architectural Support for Computer-Aided Design," panel member, SASIMI Workshop on Synthesis and Simulation, May 8-9, 1989.
33. J. White, "A Fast Multipole Algorithm for Complex 3-D Geometries," MIT VLSI Research Review, Cambridge, Massachusetts, May 22, 1989.
34. J. White, "Techniques for Switching Power Converter Simulation," Invited talk, NASECODE Conference, Dublin, Ireland, June 6-14, 1989.
35. J. White, "Waveform Relaxation for Device Transient Simulation," invited talk, SIAM meeting.

Express Cubes: Improving the Performance of k-ary n-cube Interconnection Networks¹

William J. Dally
Artificial Intelligence Laboratory
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

July 11, 1989, revised August 31, 1989

Abstract

Express cubes are k-ary n-cube interconnection networks augmented by *express channels* that provide a short path for non-local messages. An express cube combines the logarithmic diameter of an indirect network with the wire-efficiency and ability to exploit locality of a direct network. The insertion of express channels reduces the network diameter and thus the distance component of network latency. Wire length is increased allowing networks to operate with latencies that approach the physical speed-of-light limitation rather than being limited by node delays. Express channels increase wire bisection in a manner that allows the bisection to be controlled independent of the choice of radix, dimension, and channel width. By increasing wire bisection to saturate the available wiring media, throughput can be substantially increased. With an express cube both latency and throughput are wire-limited and within a small factor of the physical limit on performance. Express channels may be inserted into existing interconnection networks using *interchanges*. No changes to the local communication controllers are required.

1 Introduction

Interconnection networks are used to pass messages containing data and synchronization information between the nodes of concurrent computers [1] [2] [16] [17]. The messages may be sent between the processing nodes of a message-passing multicomputer [1] or between the processors and memories of a shared-memory multiprocessor [2].

An interconnection network is characterized by its topology, routing, and flow control [10]. The topology of a network is the arrangement of its nodes and channels into a graph. Routing determines the path chosen by a message in this graph. Flow control deals with the allocation of channel and buffer resources to a message as it travels along this path. This paper deals only with topology. Express cubes can be applied independent of routing and flow control strategies.

The performance of a network is measured in terms of its *latency* and its *throughput*. The latency of a message is the elapsed time from when the message send is initiated until the message is

¹The research described in this paper was supported in part by the Defense Advanced Research Projects Agency under contracts N00014-88K-0738 and N00014-87K-0825 and in part by a National Science Foundation Presidential Young Investigator Award with matching funds from General Electric Corporation and IBM Corporation.

completely received. Network latency is the average message latency under specified conditions. Network throughput is the number of messages the network can deliver per unit time.

Low-dimensional k -ary n -cube networks using *wormhole routing* have been shown to provide low latency and high throughput for networks that are wire-limited [4] [5] [9]. For $n \leq 3$, the k -ary n -cube topology is wire-efficient in that it makes efficient use of the available bisection width. This topology maps into the three physical dimensions in a manner that allows messages to use all of the available bandwidth along their path without ever having to double back on themselves. Also, low-dimensional k -ary n -cubes concentrate bandwidth into a few wide channels so that the component of latency due to message length is reduced. In most contemporary concurrent computers, this is the dominant component of latency. Because of their low-latency, high throughput, and affinity for implementation in VLSI, these k -ary n -cube networks with $n = 2$ or 3 have been used successfully in the design of several concurrent computers including the Ametek 2010 [17], the J-Machine [7] [8], and the Mosaic [18].

However, low-dimensional k -ary n -cube interconnection networks have two significant shortcomings:

- Because wires are short, node delays dominate wire delays and the distance related component of latency falls more than an order of magnitude short of speed-of-light limitations. In the J-Machine [7], for example, node delay is 50ns while the longest wire is 225mm and has a time-of-flight delay of 1.5ns.
- The channel width of these networks is often limited by node pin count rather than by wire bisection. For example, the J-Machine channel width is limited to 9-bits by pin count limitations. In the physical node width of 50mm, a 6-layer printed circuit board can handle over four times this channel width after accounting for through holes and local connections.

In short, many regular k -ary n -cube interconnection networks are node-limited rather than wire-limited. In these networks, node delay and pin limitations dominate wire delay and wire density limitations. The ratios of node delay to wire delays and pin density to wire density cannot be balanced in a regular k -ary n -cube.

Express cubes overcome this problem by allowing wire length and wire density to be adjusted independently of the choice of radix, k , dimension, n , and channel width, W . An express cube is a k -ary n -cube augmented by one or more levels of express channels that allow non-local messages to bypass nodes. The wire length of the express channels can be increased to the point that wire delays dominate node delays. The number of express channels can be adjusted to increase throughput until the available wiring media is saturated. This ability to balance node and wire limitations is achieved without sacrificing the wire-efficiency of k -ary n -cube networks. The number of channels traversed by a message in a hierarchical express cube grows logarithmically with distance as in a multistage interconnection network [11][19]. The express cube, however, is able to exploit locality while in a multistage network all messages must traverse the diameter of the network. With an express cube, both latency and throughput are wire limited and are within a small constant factor of the physical limit on performance.

The remainder of this paper describes the express cube topology and analyzes its performance.

Section 2 summarizes the notation that will be used throughout the paper. Section 3 introduces the express cube topology in steps. Basic express cubes (Section 3.1) reduce latency to twice the delay of a dedicated wire for messages traveling long distances. Throughput can be increased to saturate the available wiring density by adding multiple express channels (Section 3.2). With a hierarchical express cube (Section 3.3), latency for short distances, while node-limited, is within a small constant factor of the best that can be achieved by any bounded degree network. Some design considerations for express cube interchanges are discussed in Section 4.

2 Notation

The following symbols are used in this paper. They are listed here for reference.

- C , the set of channels in the network.
- D , manhattan distance traveled by a message, $|x_s - x_d| + |y_s - y_d| + |z_s - z_d|$, where the source is at (x_s, y_s, z_s) and the destination is at (x_d, y_d, z_d) .
- H hops, the number of nodes traversed by a message.
- i , number of nodes between interchanges in an express cube.
- k , the radix of the network – the length in each dimension.
- l , the number of levels of hierarchy in a hierarchical express cube.
- L , the message length in bits.
- n , the dimension of the network.
- N , the set of nodes in the network. Where it is unambiguous, N is also used for the number of nodes in the network, $|N|$.
- T_n , the latency of a node.
- T_w , the latency of a wire that connects two physically adjacent nodes.
- T_p , the pipeline period of a node.
- W , the width of a channel in bits.
- α , the ratio of node latency to wire latency, T_n/T_w .

Communication between nodes is performed by sending *messages*. A message may be broken into one or more *packets* for transmission. A packet is the smallest unit that contains routing and sequencing information. Packets contain one or more flow control digits or *flits*. A flit is the smallest unit on which flow control is performed. A flit in turn is composed of one or more physical transfer units or *phits*². A phit is W -bits, the size of the physical communication media.

An interconnection network consists of a set of nodes, N , that are connected by a set of channels, $C \subseteq N \times N$. Each channel is unidirectional and carries data from a source node to a destination node. For the purposes of this paper it is assumed that the network is bidirectional: channels occur in pairs so that $(n_1, n_2) \in C \Rightarrow (n_2, n_1) \in C$.

²There is no constraint that the physical unit of transfer, phit, must be smaller than the flow control unit, flit. It is possible to construct systems with several flits in each phit.

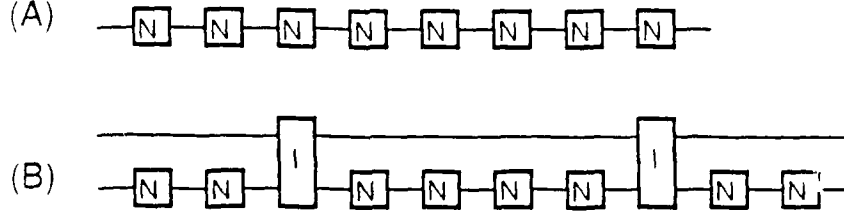


Figure 1: Insertion of express channels reduces latency: (A) A regular k -ary 1-cube network may be dominated by node delay, (B) A k -ary 1-cube with express channels reduces the node delay component of latency.

3 Express Cubes

3.1 Express Channels Reduce Latency

Figure 1 illustrates the application of express channels to a k -ary 1-cube or linear array. A regular k -ary 1-cube is shown in Figure 1A. The network is linear array of k processing nodes, labeled N, each connected to its nearest neighbors by channels of width W . The delay of a phit propagating through a node is T_n . The delay of the wire connecting two nodes is T_w . Each channel can accept a new phit every T_p . The latency of a message of length L sent distance D is

$$T_a = HT_n + DT_w + \frac{L}{W}T_p = (T_n + T_w)D + \frac{L}{W}T_p. \quad (1)$$

Message latency is composed of three components as shown in equation (1). The first component is the node latency, due to the number of hops, H . The second component is the wire latency, due to the distance D . The third component is due to message length, L . For a conventional k -ary n -cube, $H = D$ and since for most networks $T_n \gg T_w$, the node latency dominates the wire latency. Express cubes reduce the node latency by increasing wire length to reduce the number of hops, H .

An express k -ary 1-cube is shown in Figure 1B. Express channels have been added to the array by inserting an interchange, labeled I, every i nodes. An interchange is not a processing node. It performs only communication functions and is not assigned an address. Each interchange is connected to its neighboring interchanges by an additional channel of width W , the express channel. When a message arrives at an interchange it is routed directly to the next interchange if it is not destined for one of the intervening nodes. To preserve the wire-efficiency of the network, messages are never routed past their destinations on the express channels even though doing so would reduce H in many cases.

The delay, T_n , and throughput, $1/T_p$, of an interchange are assumed to be identical to those of

a node. The wire delay of the express channel is assumed to be iT_w . To simplify the following analysis, it is assumed that interchanges add no physical distance to the network. Assuming $i|D$, $H = D/i + i$ and insertion of express channels reduces the latency to

$$T_b = \left(\frac{D}{i} + i\right) T_n + T_w D + \frac{LT_p}{W}. \quad (2)$$

In the general case, an average message traversing D processing nodes travels over $H_i = (i+1)/2$ local channels to reach an interchange, $H_e = \lfloor D/i - 1/2 + 1/(2i) \rfloor$ express channels to reach the last interchange before the destination, and finally $H_f = (D - i/2 + 1/2) \bmod i$ local channels to the destination. The total number of hops is $H = H_i + H_e + H_f$ giving a latency of

$$T_b = \left(\frac{i+1}{2} + \left\lfloor \frac{D}{i} - \frac{1}{2} + \frac{1}{2i} \right\rfloor + \left((D - \frac{i}{2} + \frac{1}{2}) \bmod i\right)\right) T_n + DT_w + \frac{LT_p}{W}. \quad (3)$$

For large distances, $D \gg \alpha = T_n/T_w$, choosing $i = \alpha$ balances the node and wire delay. With this choice of i , the latency due to distance is approximately twice the wire latency, $T_D \approx 2T_w D$. The latency for large distances of large express channel network with $i = \alpha$ is within a factor of two of the latency of a dedicated manhattan wire between the source and destination³.

For small distances or large α , the i term in the coefficient of T_n in equation (2) is significant and node delay dominates. For such networks, latency is minimized by choosing $i = \sqrt{D}$ resulting in $T_D \approx 2(\sqrt{D} - 1)T_n$. The use of hierarchical express channels (Section 3.3) can further improve the latency for small distances.

3.2 Multiple Express Channels Increase Throughput to Saturate Wire Density

To first order, network throughput is proportional to wire bisection and hence wire density. If more wires are available to transmit data across the network, throughput will be increased provided that routing and flow control strategies are able to profitably schedule traffic onto these wires. Many regular network topologies, such as low-dimensional k -ary n -cubes, are unable to make use of all available wire density because of pin limitations. The wire bisection of an express cube can be controlled independent of the choice of radix, k , dimension, n , or channel width, W by adding multiple express channels to the network to match network throughput with the available wiring density.

Figure 2 shows two methods of inserting multiple express channels. Multiple express channels may be handled by each interchange as shown in Figure 2A. Alternatively, simplex interchanges can be interleaved as shown in Figure 2B.

In method A, using multiple channel interchanges, an interchange is inserted every i nodes as above and each interchange is connected to its neighbors using m parallel express channels. Figure 2A shows a network with $i = 4$ and $m = 2$. The interchange acts as a concentrator combining

³There is nothing special about the factor of two. By choosing $i = j\alpha$ the distance component of latency will be $(1 + 1/j)$ times the latency of a manhattan wire.

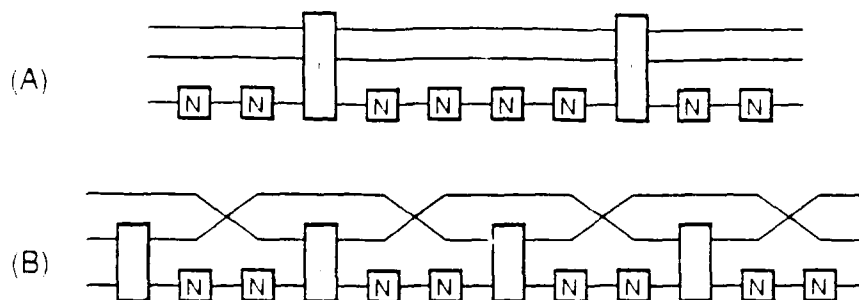


Figure 2: Multiple express channels allow wire density to be increased to saturate the available wiring media. Express channels can be added using either (A) interchanges with multiple express channels, or (B) interleaved simplex interchanges.

messages arriving on the m incoming express channels with non-local messages arriving on the local channel and concentrating these message streams onto the m outgoing express channels. This method has the advantage of making better use of the express channels since any message can route on any express channel. Flexibility in express channel assignment is achieved at the expense of higher pincount and limited expansion.

With method B, interleaving simplex interchanges, m simplex interchanges are inserted into each group of i nodes. Each interchange is connected to the corresponding interchange in the next group by a single express channel. All messages from the nodes immediately before an interchange will be routed on that interchange's express channels. Because load cannot be shared among interleaved express channels, an uneven distribution of traffic may result in some channels being saturated while parallel channels are idle. Method B has the advantage of using simple interchanges and allowing arbitrary expansion. In the extreme case of inserting an interchange between every pair of nodes the resulting topology is almost the same as the topology that would result from doubling the number of dimensions.

Both of the methods illustrated in Figure 2 have the effect of increasing the wire density (and bisection) by a factor of $m + 1$. To first order, network throughput will increase by a similar amount. There will be some degradation due to uneven loading of parallel channels.

The use of multiple express channels offsets the load imbalance between express and local channels. If traffic is uniformly distributed, the average fraction of messages crossing a point in the network on a local channel is $P_l = 2i/k$ as compared to $P_e = (k - 2i)/k$ crossing on an express channel. For large networks where $k \gg i$, the bulk of the traffic is on express channels. Increasing the number of express channels applies more of the network bandwidth where it is most needed.

Multiple express channels are an effective method of increasing throughput in networks where the channel width is limited by pinout constraints. For example, in the J-Machine the channel width, $W = 9$, is set by pin limitations⁴. The printed-circuit board technology is capable of running 80

⁴Each J-Machine node is packaged in a 168-pin PGA. The six communication channels each require 9 data bits and 6 control bits consuming 90 of these pins. Power connections use 48 pins. The remaining 30 pins are used by

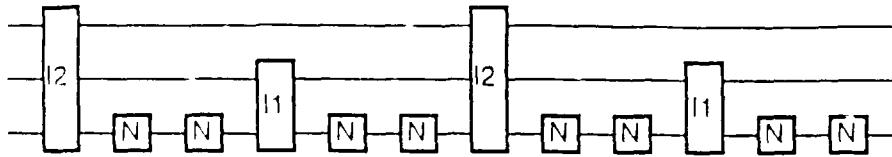


Figure 3: Hierarchical express channels reduce latency due to local routing.

wires in each dimension across the 30mm width of a node. Even with many of these wires used for local connections, four parallel 15-bit (data+control) wide channels can be easily run across each node. A multiple express channel network with $m = 3$ could use this available wire density to quadruple the throughput of the network.

3.3 Hierarchical Express Cubes Have Logarithmic Node Delay

With a single level of express channels, an average of i local channels are traversed by each non-local message. The node delay on these local channels represents a significant component of latency and causes networks with short distances, $D \leq \alpha^2$, to be node limited. Hierarchical express cubes overcome this limitation by using several levels of express channels to make node delay increase logarithmically with distance for short distances.

The use of hierarchical express channels, shown in Figure 3, reduces the latency due to node delay on local channels. With hierarchical express channels, there are l levels of interchanges. A first-level interchange is inserted every i nodes. A second-level interchange replaces every i^{th} first level interchange, every i^2 nodes. In general, a j^{th} level interchange replaces every i^{th} $j-1^{\text{st}}$ level interchange, every i^j nodes⁵. Figure 3 illustrates a hierarchical express cube with $i = 2$, $l = 2$.

A j^{th} level interchange has $j+1$ inputs and $j+1$ outputs. Arriving messages are treated identically regardless of the input on which they arrive. Messages that are destined for one of the next i nodes are routed to the local (0^{th}) output. Those remaining messages that are destined for one of the next i^2 nodes are routed to the 1^{st} output. The process continues with all messages with a destination between i^p and i^{p+1} nodes away, $0 \leq p \leq j-1$, routed to the p^{th} output. All remaining messages are routed to the j^{th} output.

A message in a hierarchical express cube is delivered in three phases: ascent, cruise, and descent. In the ascent phase, an average message travels $(i+1)/2$ hops to get to the first interchange, and $(i-1)/2$ hops at each level for a total of $H_a = (i-1)l/2 + 1$ hops and a distance of $D_a = (i^l - 1)/2$. During the cruise phase, a message travels $H_c = \lfloor (D - D_a)/i^l \rfloor$ hops on level

external memory interface and control.

⁵This construction yields a fixed-radix express cube, with radix i for each level. It is also possible to construct mixed-radix express cubes where the radix varies from level to level.

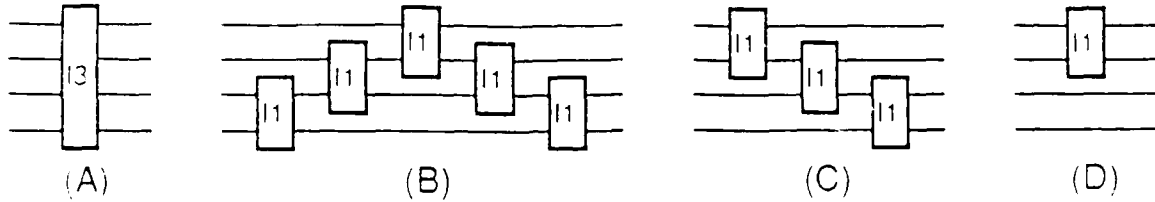


Figure 4: Hierarchical interchanges (A) a third-level interchange. (B) a third-level interchange implemented from first-level interchanges. (C,D) With a small performance penalty, ascending and/or descending interchanges can be eliminated.

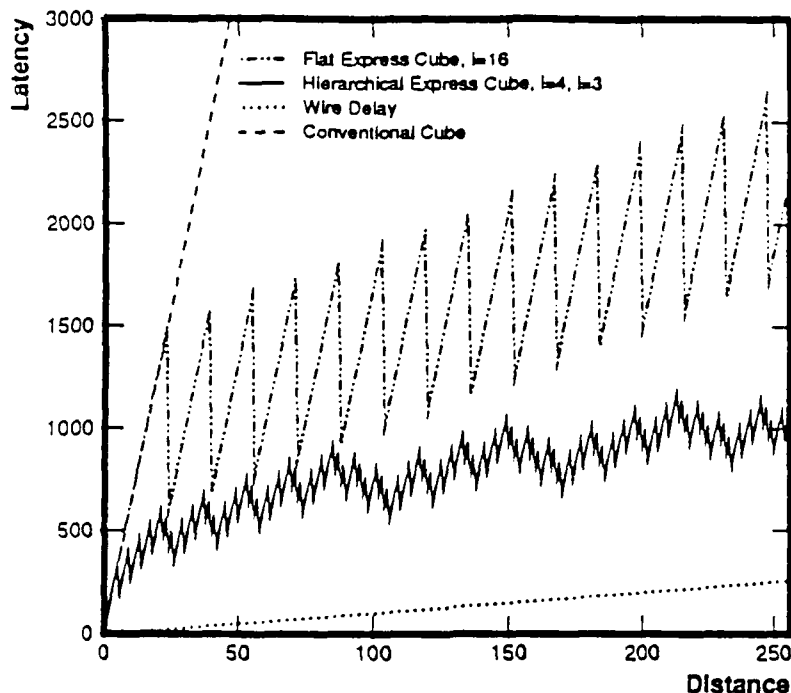
l channels for a distance of $D_c = i^l H_c$. Finally, the message descends back through the levels routing on each level, j , as long as the remaining distance is greater than i^j . For the special case where $i^l | D$, the descending message takes $H_d = (i-1)l/2 + 1$ hops for a distance of $D_d = (i^l + 1)/2$. This gives a latency of

$$T_c = \left(\frac{D}{i^l} + (i-1)l + 1 \right) T_n + T_w D + \frac{LT_p}{W}. \quad (4)$$

Choosing i and l so that $i^l = \alpha$ balances node and wire delay for large distances. With this choice, the delay due to local nodes is $(i-1)lT_n = (i-1)\log_i \alpha T_n$ which is a minimum for $i = e$. While 3 is the closest integer to e , a choice of $i = 4$ is preferred to facilitate decoding of binary addresses in interchanges, and networks with $i = 8$ or $i = 16$ may be desirable under some circumstances.

In the general case, $i^l \nmid D$, the latency of a hierarchical express cube is calculated by representing the source and destination coordinates as $h = \log_i k$ -digit radix- i numbers, $S = s_{h-1} \dots s_0$, and $D = d_{h-1} \dots d_0$. WLOG we assume that $S < D$. During the ascent phase, a message routes from S to $s_{h-1} \dots s_{l+1} 0 \dots 0$ taking $H_a = \sum_{j=0}^{l-1} ((i - s_j) \bmod i)$ hops for a distance of $D_a = \sum_{j=0}^{l-1} ((i - s_j) \bmod i) i^j$. The cruise phase takes the message $H_c = \sum_{j=l}^{h-1} (d_j - s_j) i^{j-l}$ hops for a distance of $D_c = H_c i^l$. Finally, the descent phase takes the message from $d_{h-1} \dots d_l 0 \dots 0$ to D taking $H_d = \sum_{j=0}^{l-1} d_j$ hops for a distance of $D_d = \sum_{j=0}^{l-1} d_j i^j$. For short distances the cruise phase will never be reached. The message will move from ascent to descent as soon as it reaches a node where all non-zero coordinates agree with D . The total latency for the general case is plotted as a function of distance in Figure 5.

Figure 4 shows how hierarchical interchanges can be implemented using pin-bounded modules. A level- j interchange requires $j+1$ inputs and outputs if implemented as a single module as shown for a third level interchange in Figure 4A. A level- j interchange can be decomposed into $2j-1$ level-one interchanges as shown for $j=2$ in Figure 4B. A series of $j-1$ ascending interchanges that route non-local traffic toward higher levels is followed by a top-level interchange and a series of $j-1$ descending interchanges that allow local traffic to descend. With some degradation in performance, the ascending interchanges can be eliminated as shown in Figure 4C. This change



Delay vs. Distance for Express Cubes

Figure 5: Latency as a function of distance for a hierarchical express channel cube with $i = 4$, $l = 3$, $\alpha = 64$, and a flat express channel cube with $i = 16$, $\alpha = 64$. In a hierarchical express channel cube latency is logarithmic for short distances and linear for long distances. The crossover occurs between $D = \alpha$ and $D = i\alpha \log_i \alpha$. The flat cube has linear delay dominated by T_n for short distances and by T_w for long distances.

requires extra hops in some cases as a message cannot skip levels on its way up to a high-level express channel. Each message must traverse at least one level $j - 1$ channel before being switched to a level- j channel. By restricting messages to also travel on at least one channel at each level as they descend, the descending interchanges can be eliminated as well leaving only the single top-level interchange as shown in Figure 4D.

3.4 Performance Comparison

Figure 5 shows how latency varies with distance in hierarchical and flat express cubes and compares these latencies with the latency of a conventional k -ary 1-cube and of a direct wire. These curves assume that the message source is midway between two interchanges. The latencies are normalized to units of the wire delay between adjacent nodes. The latency of a conventional k -ary 1-cube is linear with slope α while the latency of a wire is linear with slope 1.

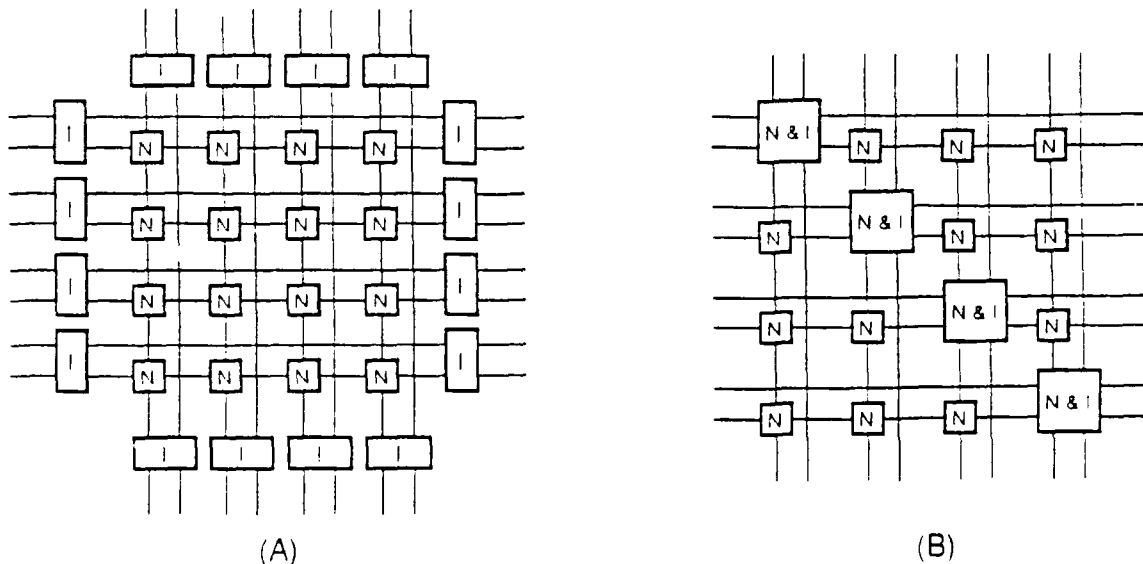


Figure 6: A multidimensional express cube may be constructed either by (A) inserting interchanges into each dimension separately, or (B) interleaving multi-dimensional interchanges into the array.

For short distances, until the first express channel is reached, a flat (non-hierarchical) express cube has the same delay as a conventional k -ary n -cube, $T_D = \alpha D$. Once the message begins traveling on express channels, latency increases linearly with slope $1 + \alpha/i$. This occurs at distance $D = 24$ in the figure. There is a periodic variation in delay around this asymptote due to the number of local channels being traversed, $D_{\text{local}} = (i + 1)/2 + ((D - i/2 + 1/2) \bmod i)$.

The hierarchical express cube has a latency that is logarithmic for short distances and linear for long distances. The latency of messages traveling a short distance, $D < \alpha$ is node limited and increases logarithmically with distance, $T_D \approx (i - 1) \log_i D T_n$. This delay is within a factor of $i - 1$ of the best that can be achieved with radix i switches. Long distance messages have a latency of $T_D \approx (1 + \alpha/i^l) T_w$. If $i^l = \alpha$, this long distance latency is approximately twice the latency of a dedicated manhattan wire. In a hierarchical network, the interchange spacing, i , can be made small, giving good performance for short distances, without compromising the delay of long distance messages which depends on the ratio α/i^l . In a flat network with a single parameter, i , it is not possible to simultaneously optimize performance for both short and long distances.

3.5 Express Channels in Many Dimensions

A multidimensional express cube may be constructed by inserting interchanges into each dimension separately as shown in Figure 6A. The figure shows part of a two-dimensional express cube with $i = 4$, $l = 1$. Interchanges have been inserted separately into the X and Y dimensions. A

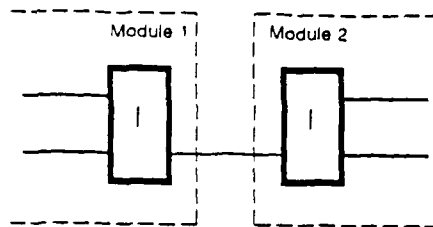


Figure 7: Interchanges allow wire density, speed, and signalling levels to be changed at module boundaries.

similar construction can be realized for higher dimensions and for hierarchical networks. With this approach interchange pin-count is minimal as each interchange handles only a single dimension. Also, the design is easy to package into modules as the interchanges are located in regular rows and columns. This approach has the disadvantage that messages must descend to local channels to switch dimensions.

An alternate construction of a multidimensional express cube is to interleave multidimensional interchanges into the array as shown in Figure 6B for $i = A, i = 1$. This approach allows messages on express channels to change dimensions without descending to a local channel. It is particularly useful in networks that use adaptive routing [13][14] as it provides alternate paths at each level of the network. The interleaved construction has the disadvantages of requiring a higher interchange pincount and being more difficult to package into modules.

3.6 Modularity

The interchanges in an express cube can be used to change wire density, speed, and signalling levels at module boundaries as shown in Figure 7. Large networks are built from many modules in a physical hierarchy. A typical hierarchy includes integrated circuits, printed circuit boards, chassis, and cabinets. Available wire density and bandwidth change significantly between levels of the hierarchy. For example, a typical integrated circuit has a wire density of 250 wires/mm per layer while a printed circuit board can handle only 2 wires/mm per layer⁶. Interchanges placed at module boundaries as shown in Figure 7 can be used to vary the number and width of express and local channels. These boundary interchanges may also convert internal module signalling levels and speeds to levels and speeds more appropriate between modules. Using express channels and boundary interchanges, the network can be adjusted to saturate the available wiring density even though this density is not uniform across the packaging hierarchy. To make use of the available bandwidth, computations running on the network must exploit locality.

⁶This integrated circuit wire density is typical of first-level metal in a 1μ CMOS process. The printed circuit wire density is for a board with 8mil wires and spaces. Both densities assume all area is available for wiring.

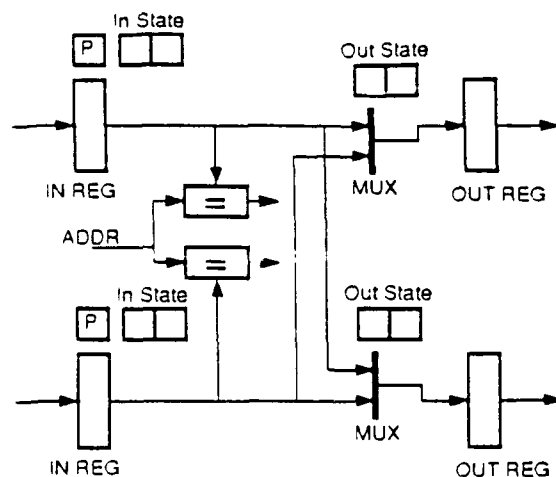


Figure 8: Block diagram of an interchange. Two multiplexors perform switching between input and output registers based on a comparison of the high address bits in a message header.

4 Interchange Design

Figure 8 shows the block diagram of a unidirectional interchange. A bidirectional interchange includes an identical circuit in the opposite direction. The basic design is similar to that of a router [15][6][3]. Two input latches hold arriving flits and two output latches hold departing flits. If additional buffering is desired, any of these latches may be replaced by a FIFO buffer. If a phit is a different size than a flit, multiplexing and demultiplexing is required between the flit buffers and the interchange pins. Associated with each output latch is a multiplexor that selects which input is routed to the latch. Routing decisions are made by comparing the address information in the head flit(s) of the message to the local address. If the destination lies within the next i nodes, the local channel is chosen, otherwise the express channel is chosen. If i is a power of two, interchanges are aligned, and absolute addresses are used in headers, the comparison can be made by checking all but the $\lceil \log_2 i \rceil$ least significant bits for equality to the local address.

The interchange state includes presence bits for each register, an input state for each input, and an output state for each output. The presence bits are used for flit-level flow control. A flit is allowed to advance only if the presence bit of its destination register is clear (no data present), or if the register is to be emptied in the same cycle. The input state bits hold the destination port and status (empty, head, advancing, blocked) of the message currently using each input. The output state consists of a bit to identify whether the output is busy and a second bit to identify which input has been granted the output. The combinational logic to maintain these state bits and control the data path is straightforward.

5 Conclusion

Express cubes are k -ary n -cubes augmented by express channels that provide a short path for non-local messages. An express cube retains the wire efficiency of a conventional k -ary n -cube while providing improved latency and throughput that are limited only by the wire delay and available wire density. For short distances, a hierarchical express cube has a latency that is within a small factor of the best that can be achieved with a bounded degree network. For long distances, the latency can be made arbitrarily close to that of a dedicated manhattan wire. Multiple express channels can be used to increase throughput to the limit of the available wire density. The express cube combines the low diameter of multistage interconnection networks with the wire efficiency and ability to exploit locality of a direct network. The result is a network with latency and throughput that are within a small factor of the physical limit.

Express channels are added to a k -ary n -cube by periodically inserting interchanges into each dimension. No modifications are required to the routers in each processing node; express channels can be added to most existing k -ary n -cube networks. Interchanges also allow wire density, speed, and signalling levels to be changed at module boundaries. An express cube can make use of all available wire density even if the wire density is non-uniform. This is often required as the wire density and speed may change significantly between levels of packaging.

Express cubes achieve their performance at the cost of adding interchanges, increasing the latency for some short-distance messages, and increasing the bisection width of the network. Each interchange adds a component to the system and increases the latency of local messages that cross an interchange but do not take the express channel by one node delay, $(T_n + T_w)$. Express channels increase the wire bisection by using available unused wiring capacity. In parts of the network that are already wire-limited the express and local channels can be combined as shown in Figure 7.

As the performance of interconnection networks approaches the limits of the underlying wiring media their range of application increases. These networks can go beyond exchanging messages between the nodes of concurrent computers to serving as a general interconnection media for digital electronic systems. For distances larger than $D' = \alpha \log \alpha$, the delay of a hierarchical express cube network is within a factor of three of that of a dedicated wire. The network may provide better performance than the wire because it is able to share its wiring resources among many paths in the network while a dedicated wire serves only a single source and destination. For distances smaller than D' , dedicated wiring offers a significant latency advantage at the cost of eliminating resource sharing.

6 Acknowledgment

I thank Charles Leiserson for pointing out the node-limited nature of most real k -ary n -cubes. Express cubes have been strongly influenced by Charles' work on fat trees [12]. I thank Steve Ward, Anant Agarwal, and Tom Knight for many helpful comments and suggestions about routing networks and their analysis. I thank Chuck Seitz for many helpful suggestions about networks, routers, and concurrent computers. I thank Andrew Chien and Michael Noakes for their careful

review of early drafts of this manuscript. Finally I thank all the members of the MIT Concurrent VLSI Architecture group for their help with and contributions to this paper.

References

- [1] Athas, William C. and Seitz, Charles L., "Multicomputers: Message-Passing Concurrent Computers," *Computer*, Vol. 21, No. 8, August 1988, pp. 9-24.
- [2] BBN Advanced Computers, Inc., *Butterfly Parallel Processor Overview*, BBN Report No. 6148, March 1986.
- [3] Dally, William J. and Seitz, Charles L., "The Torus Routing Chip," *J. Distributed Systems*, Vol. 1, No. 3, 1986, pp. 187-196.
- [4] Dally, William J., *A VLSI Architecture for Concurrent Data Structures*, Kluwer, Hingham, MA, 1987.
- [5] Dally, William J., "Wire Efficient VLSI Multiprocessor Communication Networks," *Proceedings Stanford Conference on Advanced Research in VLSI*, Paul Losleben, Ed., MIT Press, Cambridge, MA, March 1987, pp. 391-415.
- [6] Dally, William J., and Song, Paul., "Design of a Self-Timed VLSI Multicomputer Communication Controller," *Proc. International Conference on Computer Design, ICCD-87*, 1987, pp. 230-234.
- [7] Dally, William J. et.al., "The J-Machine: A Fine-Grain Concurrent Computer," *IFIP Congress*, 1989.
- [8] Dally, William J., "The J-Machine: System Support for Actors," *Actors: Knowledge-Based Concurrent Computing*, Hewitt and Agha eds., MIT Press, 1989.
- [9] Dally, William J., "Performance Analysis of k -ary n -cube Interconnection Networks," *IEEE Transactions on Computers*, to appear.
- [10] Dally, William J., "Network and Processor Architecture for Message-Driven Computing," in *VLSI and Parallel Processing*, R. Suaya and G. Birtwistle eds., Morgan Kaufmann, to appear in 1989.
- [11] Lawrie, Duncan H., "Alignment and Access of Data in an Array Processor," *IEEE Transactions on Computers*, Vol. C-24, No. 12, December 1975, pp. 1145-1155.
- [12] Leiserson, Charles E., "Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing," *IEEE Transactions on Computers*, Vol. C-34, No. 10, October 1985, pp. 892-900.
- [13] Mailhot, John, *A Comparative Study of Routing and Flow Control Strategies in k -ary n -cube Networks*, S.B. Thesis, Massachusetts Institute of Technology, May 1988.
- [14] Ngai, John, *A Framework for Adaptive Routing in Multicomputer Networks*, Ph.D. Thesis, Caltech Computer Science Technical Report, Caltech-CS-TR-89-09, May 1989.
- [15] Nuth, P.R., *Router Protocol*, MIT Concurrent VLSI Architecture Memo No. 23, February 1989.
- [16] Seitz, Charles L., "The Cosmic Cube", *Comm. ACM*, Vol. 28, No. 1, Jan. 1985, pp. 22-33.
- [17] Seitz, C.L., et. al., "The Architecture and Programming of the Ametek Series 2010 Multicomputer," *Proc. Third Conference on Hypercube Concurrent Computers and Applications*, ACM, Jan. 1988, pp. 33-37.
- [18] Seitz, C.L., et. al., *Submicron Systems Architecture Project Semiannual Technical Report*, Caltech Computer Science Technical Report, Caltech-CS-TR-88-18, November 1988, p. 2 and pp. 11-12.
- [19] Wu, Chuan-Lin, and Feng Tse-Yun, "On a Class of Multistage Interconnection Networks," *IEEE Transactions on Computers*, Vol. C-29, No. 8, August 1980, pp. 694-702.

Priority Arbitration with Busses

(Extended Abstract)

Shlomo Kipnis

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

October 16, 1989

Abstract

This paper explores priority arbitration schemes that employ busses to arbitrate among n modules in a digital system. We focus on distributed mechanisms that employ m busses, for $\lg n \leq m \leq n$, and use asynchronous combinational arbitration logic. A widely used distributed asynchronous mechanism is the *binary arbitration* scheme, which with $m = \lg n$ busses arbitrates in $t = \lg n$ units of time. We present a new asynchronous scheme — *binomial arbitration* — that by using $m = \lg n + 1$ busses reduces the arbitration time to $t = \frac{1}{2} \lg n$. Extending this result, we present the *generalized binomial arbitration* scheme that achieves a bus-time tradeoff of the form $m = \Theta(tn^{1/t})$ between the number of arbitration busses m and the arbitration time t (in units of bus-settling delay), for values of $1 \leq t \leq \lg n$ and $\lg n \leq m \leq n$. Our schemes are based on a novel analysis of *data-dependent delays* and generalize the two known schemes: *linear arbitration*, which with $m = n$ busses achieves $t = 1$ time, and *binary arbitration*, which with $m = \lg n$ busses achieves $t = \lg n$ time. Most importantly, our schemes can be adopted with no changes to existing hardware and protocols; they merely involve selecting a *good* set of priority arbitration codewords.

Keywords: arbitration, arbitration priorities, asynchronous arbitration, binary arbitration, binomial arbitration, busses, bus-settling delay, combinational logic, data-dependent delays, generalized binomial arbitration, linear arbitration, open-collector busses, priority arbitration, resource tradeoff, wired-OR.

1 Introduction

In many electronic systems there are situations where several modules wish to use a common resource simultaneously. Examples include microprocessor systems where a decision is required concerning which of several interrupts to service first, multiprocessor environments where several processors wish to use some device concurrently, and data communication networks with shared media. To resolve conflicts, an arbitration mechanism is required that grants the resource to one module at a time.

Numerous arbitration mechanisms have been developed, including daisy chains, priority circuits, polling, token passing, and carrier sense protocols, to name a few (see [5, 6, 10, 14, 18, 19, 22, 26]). In this paper we focus on distributed *priority arbitration* mechanisms, where contention is resolved using predetermined module priorities and the arbitration process is carried out in a distributed manner at all the system modules. In many modern systems, and especially in multiprocessor environments and data communication networks, distributed priority arbitration is the preferred mechanism.

Many distributed arbitration mechanisms employ a collection of *arbitration busses* to implement priority arbitration. To this end, each module is assigned a unique *arbitration priority*, which is an encoding of its name. An arbitration protocol determines the logic values that a module applies to the busses, based on the module's arbitration priority and on logic values on other busses. After some delay, the settled logic values on the busses uniquely identify the contending module with the highest priority. In particular, the *asynchronous binary arbitration* scheme, developed by Taub [23], gained popularity and is used in many modern bus systems, such as Futurebus [7, 25], M3-bus [9], S-100 bus [13, 24], Multibus-II [14], Fastbus [15], and Nubus [28]. Other priority arbitration mechanisms that employ busses are described in [5, 6, 10, 12, 17, 18, 19, 22, 26].

The asynchronous binary arbitration scheme arbitrates among n modules in $t = \lg n$ units of time, using $m = \lg n$ open-collector (wired-OR) arbitration busses.¹ The technology of open-collector busses is such that the default logic value on a bus is 0, unless at least one module applies a 1 to it, in which case it becomes a 1. Open-collector busses, thus, OR together the logic values applied to them, with some time delay called *bus-settling delay*. In asynchronous binary arbitration, each module is assigned a unique $(\lg n)$ -bit arbitration priority. When arbitration begins, competing modules apply their arbitration priorities to the $m = \lg n$ busses, each bit on a separate bus; the result being the bitwise OR of their arbitration priorities. As arbitration progresses, each competing module monitors the busses and disables its drivers according to the following rule: if the module is applying a 0 (that is, not applying a 1) to a particular bus but detects that the bus is carrying a 1 (applied by some other module), it ceases to apply all its bits of lower significance. Disabled bits are re-enabled should the condition cease to hold. The effect of this rule is that the arbitration proceeds in $\lg n$ stages from the most significant bit to the least significant bit. Each stage consists of resolving another bit of the highest competing binary priority, which leads to a worst-case arbitration time of $t = \lg n$ (in units of bus-settling delay).

¹Throughout this paper we count only arbitration busses that are used for encoding the priorities. Several additional control busses are used by all schemes and are therefore not counted.

	Stage 1					Stage 2					Stage 3					Stage 4				
	c_2	c_5	c_9	c_{10}	OR	c_2	c_5	c_9	c_{10}	OR	c_2	c_5	c_9	c_{10}	OR	c_2	c_5	c_9	c_{10}	OR
Bus b_3	0	0	1	1	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1	1
Bus b_2	0	1	0	0	1	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0
Bus b_1	1	0	0	1	1	1	0	0	1	0	1	0	0	1	1	1	0	0	1	1
Bus b_0	0	1	1	0	1	0	1	1	0	0	0	1	1	0	1	0	1	1	0	0

Figure 1: Asynchronous binary arbitration process with 4 busses. The competing modules are c_2 , c_5 , c_9 , and c_{10} , with corresponding arbitration priorities 0010, 0101, 1001, and 1010. Bits in shaded regions are not applied to the busses. The process takes 4 stages.

For example, consider a system of $n = 16$ modules that uses $m = \lg 16 = 4$ arbitration busses, with the 16 arbitration priorities consisting of all the 4-bit codewords $\{0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111\}$. Figure 1 outlines an asynchronous binary arbitration process among four such modules c_2 , c_5 , c_9 , and c_{10} , with corresponding arbitration priorities 0010, 0101, 1001, and 1010. The arbitration process begins by bitwise ORing the four arbitration priorities. After one unit of bus-settling delay (stage 1), bus b_3 settles to the value 1, where it will remain for the duration of the arbitration. By the above rule, each of modules c_2 and c_5 disables its last three bits. In the meantime, however, each of modules c_9 and c_{10} disables its last two bits, because of the 1 on bus b_2 . At the end of stage 2, bus b_2 settles to the value 0, where it will remain for the rest of the process. As a result, modules c_9 and c_{10} now re-enable their low order bits (stage 3), which results in bus b_1 settling to a 1 at the end of stage 3. Finally, in stage 4, module c_9 ceases to apply its last bit, because of the 1 it detects on bus b_1 , which results in bus b_0 settling to a 0 at the end of stage 4. This arbitration process required $t = \lg 16 = 4$ stages to complete.

In this paper we show that the asynchronous binary arbitration scheme can in fact be improved. We introduce the new *asynchronous binomial arbitration* scheme, that uses one more arbitration bus in addition to the $\lg n$ busses of binary arbitration, but, most surprisingly, reduces the arbitration time to $\frac{1}{2} \lg n$. In asynchronous binomial arbitration, we use $(\lg n + 1)$ -bit codewords as arbitration priorities and follow the same arbitration protocol of asynchronous binary arbitration. Our binomial arbitration scheme guarantees fast arbitration by employing certain codewords that exhibit small *data-dependent delays* during arbitration processes. For example, by using the following set of 5-bit codewords $\{00000, 00001, 00010, 00011, 00100, 00110, 00111, 01000, 01100, 01110, 01111, 10000, 11000, 11100, 11110, 11111\}$ as arbitration priorities, we can arbitrate among 16 modules using 5 busses in at most 2 stages. Figure 2 outlines an asynchronous binomial arbitration process among four such modules c_1 , c_6 , c_{11} , and c_{12} , with corresponding arbitration priorities 00001, 00111, 10000, and 11000 from the above set, that completes in 2 stages. It turns out that for any subset of the above 16 codewords, the corresponding arbitration process takes at most 2 stages. In Section 3, we show how to design a good set of codewords for general values of n by using *binomial codes* as arbitration priorities

	Stage 1					Stage 2				
	c_1	c_6	c_{11}	c_{12}	OR	c_1	c_6	c_{11}	c_{12}	OR
Bus b_4	0	0	1	1	1	0	0	1	1	1
Bus b_3	0	0	0	1	1	0	0	0	1	1
Bus b_2	0	1	0	0	1	0	1	0	0	0
Bus b_1	0	1	0	0	1	0	1	0	0	0
Bus b_0	1	1	0	0	1	1	1	0	0	0

Figure 2: Asynchronous binomial arbitration process with 5 busses. The competing modules are c_1 , c_6 , c_{11} , and c_{12} , with corresponding arbitration priorities 00001, 00111, 10000, and 11000. Bits in shaded regions are not applied to the busses. The process takes 2 stages.

The remainder of this paper explores priority arbitration schemes that employ busses to arbitrate among n modules. In Section 2 we discuss priority arbitration and formally define the asynchronous model of priority arbitration with busses. Section 3 describes the two known asynchronous schemes: *linear arbitration* and *binary arbitration*, and presents our new asynchronous *binomial arbitration* scheme, which with $m = \lg n + 1$ busses arbitrates in $t = \frac{1}{2} \lg n$ units of time. In Section 4 we extend binomial arbitration and present the *generalized binomial arbitration* scheme that achieves a spectrum of bus-time tradeoff of the form $m = \Theta(tn^{1/t})$, between the number of arbitration busses m and the arbitration time t , for values of $1 \leq t \leq \lg n$ and $\lg n \leq m \leq n$. The established bus-time tradeoff is of great practical interest, enabling system designers to achieve a desirable balance between amount of hardware and speed. We present a variety of extensions to the results of this paper in Section 5.

2 Asynchronous Priority Arbitration with Busses

In this section we discuss priority arbitration and formally define the asynchronous model of priority arbitration with busses. The definitions in this section model typical implementations of asynchronous priority arbitration mechanisms that employ busses.

Arbitration is the process of selecting one module from a set of contending modules. In asynchronous priority arbitration with busses, each module is assigned a unique arbitration priority — an encoding of its name — which is used in determining logic values to apply to the busses during arbitration. An arbitration protocol determines the logic values that a competing module applies to the busses based on the module's arbitration priority and potentially also on logic values on other busses. The beginning of an arbitration process is identified by a system-wide signal, usually called REQUEST or ARBITRATE. The resolution of an arbitration process is the collection of settled logic values on the busses at the end of the process, which should uniquely identify the competing module having the highest arbitration priority.

Throughout this paper we use the following notations and assumptions. The set $C = \{c_0, c_1, \dots, c_{n-1}\}$ denotes the n system modules (chips), which we assume are indexed in increasing order of priority. The m open-collector (wired-OR) arbitration busses are denoted by $B = \{b_0, b_1, \dots, b_{m-1}\}$, where the busses are indexed in increasing order of significance (to be elaborated later). The set $P = \{p_0, p_1, \dots, p_{n-1}\}$ consists of n distinct arbitration priorities, with p_i being the arbitration priority of module c_i . Arbitration priorities are only a convenient mechanism of encoding the modules' names, and in many asynchronous schemes arbitration priorities are m -bit vectors that competing modules apply to the m busses during arbitration. When necessary, we denote the bits of an arbitration priority p by $p^{(0)}, p^{(1)}, p^{(2)}, \dots$ in order of increasing significance. We assume that each module is connected to all busses and can thus read from and potentially write to any bus. All modules follow the same arbitration protocol in interfacing with the busses and reaching conclusions concerning the arbitration process. Finally, we assume that only competing modules apply logic values to the busses; noncompeting modules do not interfere with the busses. All our assumptions are standard design practice in many systems.

In asynchronous priority arbitration with busses, we restrict the arbitration process to be purely combinational by requiring that the arbitration logic on all the modules together with the arbitration busses form an acyclic circuit. Using combinational logic with asynchronous feedback paths may introduce race conditions and metastable states, which can defer arbitration indefinitely (see [1, 20, 21]). The acyclic nature of the arbitration logic imposes a partial order on the busses, which can be extended to a linear order. The significance of the linear order on the busses is that logic values on higher indexed busses can be used to determine logic values of lower indexed busses but not vice versa. We formalize this idea in the following definition of an acyclic arbitration protocol.

Definition 1 Let P be a set of arbitration priorities. An *acyclic arbitration protocol* of size m for P is a sequence $F = \langle f_{m-1}, \dots, f_1, f_0 \rangle$ of m functions, $f_j : P \times \{0, 1\}^{m-j-1} \rightarrow \{0, 1\}$, for $j = 0, 1, \dots, m-1$.

In asynchronous priority arbitration with busses, every module has arbitration circuitry that implements the same acyclic arbitration protocol, but with the module's arbitration priority as a parameter. The m arbitration busses are ordered from b_{m-1} down to b_0 , in accordance with the acyclic nature of the circuit. Informally, function f_j takes an arbitration priority $p \in P$ and $m-j-1$ bit values on the highest $m-j-1$ busses b_{m-1} through b_{j+1} , and determines the bit value that a competing module c with arbitration priority p applies to bus b_j , for $j = 0, 1, \dots, m-1$. An arbitration process among several contending modules consists of the competing modules applying logic values to the m busses according to the acyclic arbitration protocol of size m .

Measuring the arbitration time of asynchronous mechanisms is somewhat problematic. We follow a standard approach taken in many bus systems (see [6, 10, 11, 14, 16, 24, 25]) and measure the arbitration time in units of bus-settling delay. Bus-settling delay, T_{bus} , is the time it takes for a bus to settle to a stable logic value, once its drivers have stabilized, which includes the delays introduced by the logic gates driving the bus, the bus propagation delay, and any additional time required to resolve transient effects such as the wired-OR

glitch. In effect, we model an open-collector bus as an OR gate with delay T_{bus} , the time it takes for the output of the gate to stabilize on a valid logic value, once its inputs have reached their final values. An arbitration process is modeled as a sequence of stages, each taking T_{bus} time, and the arbitration time is defined as the number of stages it takes until all busses stabilize. This approach models the situation in many bus systems rather accurately. (More discussion of measuring the arbitration time in units of bus-settling delay is deferred until Section 5.)

We next formally define the notion of an arbitration process of an acyclic arbitration protocol on a set of competing arbitration priorities. We characterize the arbitration process by the collection of the logic values on the m busses at the end of each computation stage. We use $v_j[l]$ to denote the logic value on bus b_j at the end of the l th computation stage, for $j = 0, 1, \dots, m-1$ and $l = 0, 1, \dots$. Without loss of generality, we assume that an arbitration process begins with all busses being in logic value 0.

Definition 2 Let P be a set of arbitration priorities, F be an acyclic arbitration protocol of size m for P , and $Q \subset P$ be a set of competing arbitration priorities. The *arbitration process* of F on Q is the successive evaluation of

$$\begin{aligned} v_j[0] &= 0, \\ v_j[l+1] &= \bigvee_{p \in Q} f_j(p, v_{m-1}[l], \dots, v_{j+1}[l]), \end{aligned}$$

for $j = 0, 1, \dots, m-1$ and $l = 0, 1, \dots$. We say that the arbitration process takes t stages if $t \geq 0$ is the smallest integer for which $v_j[t] = v_j[t+1]$, for $j = 0, 1, \dots, m-1$. The *resolution* of the arbitration process is the sequence of values $(v_{m-1}[t], \dots, v_1[t], v_0[t])$.

Definition 2 characterizes an arbitration process as a successive application of the acyclic arbitration protocol F to the set of competing arbitration priorities Q and the current state of the m busses. The arbitration process terminates when no more changes in the state of the busses occur, at which point a resolution is reached. It is relatively easy to verify that any arbitration process of an acyclic arbitration protocol F of size m takes at most m stages. This is the case because at each computation stage of an arbitration process, at least one more bus stabilizes on its final value.

A better upper bound for the number of stages taken by arbitration processes is given by the depth of the acyclic arbitration protocol. As discussed above, the acyclic nature of the arbitration logic imposes a partial order on the busses. We can therefore statically partition the m busses into d levels, such that the computation for a bus in a certain level only uses the values of busses in previous levels. More formally, given an acyclic arbitration protocol F of size m , we simultaneously partition the m functions of F into d nonempty disjoint sets F_0, F_1, \dots, F_{d-1} , and the m busses of B into d corresponding sets B_0, B_1, \dots, B_{d-1} , with $f_j \in F_h$ if and only if $b_j \in B_h$, for $0 \leq j \leq m-1$, and $0 \leq h \leq d-1$. The partition must have the property that the computation of a function $f_j \in F_h$ depends only on the arbitration priorities and on values of busses in sets B_0, B_1, \dots, B_{h-1} . The *depth* of an acyclic arbitration protocol F of size m is defined as the smallest d , for which a partition as above exists. The depth of an acyclic arbitration protocol is never greater

than its size. The next theorem shows that any acyclic arbitration protocol of depth d reaches a resolution after at most $t = d$ computation stages.

Theorem 1 *Let P be a set of arbitration priorities, F be an acyclic arbitration protocol of size m for P , and d be the depth of F . Then, for any subset $Q \subset P$ of competing arbitration priorities, the arbitration process of F on Q takes at most d stages.*

Proof. By induction on d , the depth of the acyclic arbitration protocol F .

Base case: $d = 0$. For depth $d = 0$, there are no arbitration busses and the claim holds immediately for arbitrary Q .

Inductive case: $d > 0$. Given an acyclic arbitration protocol $F = \langle f_{m-1}, \dots, f_1, f_0 \rangle$ of size m and depth d for P , we can partition $F = \cup_{h=0}^{d-1} F_h$ and $B = \cup_{h=0}^{d-1} B_h$ as above. Without loss of generality, we assume that the last level consists of the r functions and busses with indices $0, 1, \dots, r-1$. The first $d-1$ levels of F constitute an acyclic arbitration protocol $F' = \cup_{h=0}^{d-2} F_h = \langle f_{m-1}, \dots, f_{r+1}, f_r \rangle$ of size $m-r$ and depth $d-1$ for P . By induction, the arbitration process of F' on Q takes at most $d-1$ stages. That is, for any $r \leq j \leq m-1$ and $l \geq d-1$, we have $v_j[l] = v_j[d-1]$. In addition, according to the acyclic arbitration protocol F , we also have that for any $0 \leq i \leq r-1$ and $k \geq d > 0$,

$$\begin{aligned} v_i[k] &= \bigvee_{p \in Q} f_i(p, v_{m-1}[k-1], \dots, v_r[k-1]) \\ &= \bigvee_{p \in Q} f_i(p, v_{m-1}[d-1], \dots, v_r[d-1]) \\ &= v_i[d], \end{aligned}$$

because the d th level depends only on busses b_{m-1} down to b_r and because $k-1 \geq d-1$. This proves that the arbitration process takes at most d stages. ■

Theorem 1 shows that the number of stages that an arbitration process takes is bounded by the depth of the acyclic arbitration protocol F . This bound represents a standard *static* approach in the analysis of delays in digital circuits, namely, that of counting the number of gates on the longest path from the inputs to the outputs. In this paper, however, we introduce and use a novel *dynamic* approach of bounding the number of stages that an arbitration process takes by a careful analysis of the *data-dependent delays* experienced in the arbitration circuits. In doing so, we exhibit arbitration schemes that guarantee termination of any arbitration process in a circuit of size and depth m after a fixed number of stages t , for values of $0 \leq t \leq m$.

To complete the definition of asynchronous priority arbitration schemes, we need to introduce the notion of an interpretation function. Suppose we have a set of arbitration priorities P and an acyclic arbitration protocol F of size m for P . An *interpretation function* for P and F is a function $\text{WIN} : \{0, 1\}^m \rightarrow P$, such that for any $Q \subset P$, with $p \in Q$ being the highest arbitration priority in Q and $\langle v_{m-1}, \dots, v_1, v_0 \rangle$ being the resolution of the arbitration process of F on Q , we have $\text{WIN}(v_{m-1}, \dots, v_1, v_0) = p$. Informally, WIN interprets the resolution of any arbitration process of F by identifying the highest competing arbitration priority. We are now ready to define an asynchronous priority arbitration scheme for n modules, m busses, and t stages.

Definition 3 An *asynchronous priority arbitration scheme* for n modules, m busses, and t stages is a triplet $\mathcal{A}(n, m, t) = \langle P, F, \text{WIN} \rangle$, where

- P is a set of n arbitration priorities;
- F is an *acyclic arbitration protocol* of size m for P ;
- WIN is an *interpretation function* for P and F ;

such that for any $Q \subseteq P$, the arbitration process of F on Q takes at most t stages.

Definition 3 emphasizes the role of the arbitration priorities, which are just a mechanism to distinguish between different modules. It will become apparent, however, that careful design of the codewords used as arbitration priorities has a significant impact on the arbitration time. In the next Section, for example, we demonstrate that by using the set of $(\lg n + 1)$ -bit *binomial codes* as arbitration priorities, we can achieve an arbitration time of $t = \frac{1}{2} \lg n$.

3 Asynchronous Priority Arbitration Schemes

In this section we first use our framework to describe two commonly used asynchronous priority arbitration schemes: *linear arbitration*, which with $m = n$ busses arbitrates in time $t = 1$, and *binary arbitration*, which with $m = \lg n$ busses arbitrates in time $t = \lg n$. We then present our new asynchronous scheme, *binomial arbitration*, which with $m = \lg n + 1$ busses arbitrates in time $t = \frac{1}{2} \lg n$.

The Asynchronous Linear Arbitration Scheme

This scheme uses $m = n$ busses and arbitrates among n modules in $t = 1$ stages. To arbitrate, contending module c_i applies a 1 to bus b_i , for $0 \leq i \leq n - 1$, and does not interfere with other busses. This translates to module c_i having an n -bit arbitration priority p_i , such that $p_i^{(j)} = 1$ if $i = j$ and $p_i^{(j)} = 0$ otherwise. After $t = 1$ units of time, all the busses stabilize on their final values, and the module with a 1 on the bus with the highest priority is recognized as the winner. This scheme can also be implemented with tri-state busses, since at most one module writes to any given bus. The scheme is also known as *decoded arbitration* and is used in a number of bus systems and interrupt arbitration mechanisms (see [10, 12, 18, 26]).

Formally, we define this scheme as $\text{LINEAR}(n, n, 1) = \langle P, F, \text{WIN} \rangle$, where

- $P = \{p_i = 0^{n-i-1} 1 0^i : \text{for } i = 0, 1, \dots, n - 1\}$.
- $F = \langle f_{n-1}, \dots, f_1, f_0 \rangle$, where $f_j(p, v_{n-1}, \dots, v_{j+1}) = p^{(j)}$, for $j = 0, 1, \dots, n - 1$.
- $\text{WIN}(0^k 1 \alpha) = 0^k 1 0^{n-k-1} = p_{n-k-1}$, for $0 \leq k \leq n - 1$ and any $\alpha \in \{0, 1\}^{n-k-1}$.

Notice that although the size of the acyclic arbitration protocol of **LINEAR** is $m = n$, its depth is only $d = 1$, which according to Theorem 1 shows that the asynchronous linear arbitration scheme takes at most $t = 1$ stages to arbitrate.

The Asynchronous Binary Arbitration Scheme

This scheme uses $m = \lceil \lg n \rceil$ busses and arbitrates among n modules in $t = \lceil \lg n \rceil$ stages. The arbitration priority p_i of module c_i is the binary representation of i , for $0 \leq i \leq n - 1$. To arbitrate, contending module c drives its binary priority p onto the m busses, from $p^{(m-1)}$ (the most significant bit of p) onto bus b_{m-1} , down to $p^{(0)}$ (the least significant bit of p) onto bus b_0 ; the result being the bitwise OR of the binary priorities of the competing modules. During arbitration, each competing module c monitors the busses and disables its drivers according to the following rule: let $p^{(l)}$ be the l th bit of the binary priority p , and let v_l be the binary value observed on bus b_l , for $0 \leq l \leq m - 1$. Then if $p^{(l)} = 0$ and $v_l = 1$, module c disables all its bits $p^{(j)}$ for $j < l$. Disabled bits are re-enabled should the condition cease to hold. After $t = \lceil \lg n \rceil$ units of time, all the busses stabilize on their final values, and the module whose arbitration priority appears on the busses is the winner. This scheme was developed by Taub [23], and is also known as encoded arbitration (see [6, 10, 14, 24, 25]).

Formally, we define this scheme $\text{BINARY}(n, \lceil \lg n \rceil, \lceil \lg n \rceil) = \langle P, F, \text{WIN} \rangle$ as follows. For simplicity of notation we use $m = \lceil \lg n \rceil$.

- $P = \{p_i = \epsilon_{m-1} \cdots \epsilon_1 \epsilon_0 : \text{where } \epsilon_{m-1} \cdots \epsilon_1 \epsilon_0 \text{ is the binary representation of } i, \text{ for } i = 0, 1, \dots, n - 1\}$.
- $F = \langle f_{m-1}, \dots, f_1, f_0 \rangle$, where

$$f_j(p, v_{m-1}, \dots, v_{j+1}) = \begin{cases} 0 & \text{if } \bigvee_{l=j+1}^{m-1} (p^{(l)} = 0 \wedge v_l = 1) \\ p^{(j)} & \text{otherwise} \end{cases}.$$

for $j = 0, 1, \dots, m - 1$.

- $\text{WIN}(\alpha) = \alpha$, for any $\alpha \in \{0, 1\}^m$.

Notice that the size m and the depth d of the acyclic arbitration protocol of **BINARY** are equal, specifically $m = d = \lceil \lg n \rceil$. This can be verified by noticing that the computation for each bus b_j , where $0 \leq j \leq m - 1$, takes into account values on busses b_l , for $j < l \leq m - 1$. This implies, according to Theorem 1, that the asynchronous binary arbitration scheme takes at most $t = \lceil \lg n \rceil$ stages to arbitrate. On the other hand, it has been shown in [2, 10, 11, 24, 25, 27] that there are examples where a binary arbitration process takes exactly $\lceil \lg n \rceil$ stages. These examples consist of arbitrating among *bad* subsets of arbitration priorities, where at each stage the binary value of exactly one more bit of the highest competing binary priority is resolved. Our asynchronous binomial arbitration scheme, presented next, guarantees fast arbitration by employing certain codewords that exhibit small data-dependent delays.

The Asynchronous Binomial Arbitration Scheme

This scheme uses $m = \lceil \lg n + 1 \rceil$ busses to arbitrate among n modules in $t = \lceil \frac{1}{2} \lg n \rceil$ stages. This scheme's acyclic arbitration protocol and interpretation function are identical to those of the binary arbitration scheme, and thus the same hardware can be used. The only difference is that *binomial codes* are used as arbitration priorities rather than all the 2^m possible m -bit codewords of binary arbitration. Alternatively, with m busses, this scheme can arbitrate among 2^{m-1} modules in $t = \lceil \frac{1}{2}(m-1) \rceil$ stages. We next describe the binomial codes and begin by defining the interval-number of a binary codeword.

Definition 4 The *interval-number* of a binary codeword p is the number of intervals of consecutive 1's or 0's that it contains, disregarding leading 0's.

Thus, for example, the interval-number of 001011 is 3, the interval-number of 0000 is 0, and the interval-number of 10101010 is 8. In general, an m -bit binary codeword p with interval-number r , has the form $p = 0^{m_0}1^{m_1}0^{m_2}1^{m_3} \dots \delta^{m_r}$, where $\delta \in \{0,1\}$; $m_0 \geq 0$; $m_j > 0$ for $1 \leq j \leq r$; and $\sum_{j=0}^r m_j = m$. We next define the binomial codes of length m .

Definition 5 The set of *binomial codes* of length m , denoted by $D(m)$, is the set of all the m -bit binary codewords that have interval-number at most $\lceil \frac{1}{2}(m-1) \rceil$.

The binomial codes of length m are in fact all the m -bit codewords, that, after deleting leading 0's have at most $\lceil \frac{1}{2}(m-1) \rceil$ intervals of consecutive 1's or 0's. For example, the binomial codes of length 4 is $D(4) = \{0000, 0001, 0010, 0011, 0100, 0110, 0111, 1000, 1100, 1110, 1111\}$, consisting of 11 codewords that have interval-number at most 2. As another example, the binomial codes that were used in the introduction are $D(5) = \{00000, 00001, 00010, 00011, 00100, 00110, 00111, 01000, 01100, 01110, 01111, 10000, 11000, 11100, 11110, 11111\}$, consisting of the 16 codewords of length 5 with interval-number at most 2. For general values of m , Corollary 3 in Section 4 shows that there are at least 2^{m-1} binomial codes of length m . By taking $m = \lceil \lg n + 1 \rceil$, this translates to at least $2^{\lceil \lg n + 1 \rceil - 1} \geq n$ binomial codes, which means that there are enough arbitration priorities for n modules.

Formally, we define this scheme $\text{BINOMIAL}(n, \lceil \lg n + 1 \rceil, \lceil \frac{1}{2} \lg n \rceil) = \langle P, F, \text{WIN} \rangle$ as follows. We use $m = \lceil \lg n + 1 \rceil$ and $t = \lceil \frac{1}{2} \lg n \rceil$ for simplicity of notation.

- $P = D(m)$.
- $F = \langle f_{m-1}, \dots, f_1, f_0 \rangle$, where

$$f_j(p, v_{m-1}, \dots, v_{j+1}) = \begin{cases} 0 & \text{if } \bigvee_{l=j+1}^{m-1} (p^{(l)} = 0 \wedge v_l = 1) \\ p^{(j)} & \text{otherwise} \end{cases},$$

for $j = 0, 1, \dots, m-1$.

- $\text{WIN}(\alpha) = \alpha$, for any $\alpha \in \{0,1\}^m$.

It remains to show that the asynchronous binomial arbitration scheme indeed arbitrates among n modules in at most $t = \lceil \frac{1}{2} \lg n \rceil$ stages. Notice that a standard static analysis of the arbitration circuitry, as given for example in Theorem 1, does not give the desired result, since both the size and the depth of the acyclic arbitration protocol F of binomial arbitration are $m = d = \lceil \lg n + 1 \rceil$. In Section 4, we use a novel dynamic approach of analyzing the data-dependent delays experienced in arbitration processes, and prove the correctness of our scheme as a special case of our generalized binomial arbitration scheme.

4 Generalized Binomial Arbitration

In this section we extend the ideas of the asynchronous binomial arbitration scheme of Section 3 by presenting the *generalized binomial arbitration* scheme that with m busses and in at most t stages, arbitrates among $n = \sum_{l=0}^t \binom{m}{l}$ modules. By Stirling's approximation, the asymptotic bus-time tradeoff of the generalized binomial arbitration scheme is approximately $m = \frac{1}{2} t n^{1/t}$. This bus-time tradeoff is of great practical interest, enabling system designers to achieve a desirable balance between amount of hardware and speed. The performance of the generalized binomial arbitration scheme is based on an analysis of data-dependent delays.

We first define the set of generalized binomial codes of length m and diversity r .

Definition 6 The set of *generalized binomial codes* of length m and diversity r , denoted by $G(m, r)$, is the set of all m -bit binary codewords that have interval-number at most r .

Generalized binomial codes serve as arbitration priorities in the generalized binomial arbitration scheme. The next lemma determines the cardinality of the set of the generalized binomial codes of length m and diversity r .

Lemma 2 The set $G(m, r)$ contains $\sum_{l=0}^r \binom{m}{l}$ distinct codewords.

Proof. To simplify the counting, we take all the codewords in $G(m, r)$ and append a 0 at their beginning. This results in a set of $(m+1)$ -bit words, that begin with a 0 and have at most r switching points from a consecutive interval of 0's to a consecutive interval of 1's and vice versa. The number of such words is $\sum_{l=0}^r \binom{m}{l}$, since there are exactly that many possibilities of choosing at most r switching points out of m possible positions. ■

Corollary 3 There are at least 2^{m-1} binomial codes of length m .

Proof. By our notation, the set of binomial codes of length m , $D(m)$, is defined by $D(m) = G(m, \lceil \frac{1}{2}(m-1) \rceil)$. According to Lemma 2, we have

$$|D(m)| = \sum_{l=0}^{\lceil \frac{1}{2}(m-1) \rceil} \binom{m}{l}.$$

The sum includes the first $\left\lceil \frac{1}{2}(m-1) \right\rceil + 1$ binomial coefficients, which constitute at least a half of all the $m+1$ binomial coefficients. The partial sum is therefore at least a half of the full sum, which is 2^m . We therefore conclude that $|D(m)| \geq \frac{1}{2} \cdot 2^m = 2^{m-1}$. ■

The Asynchronous Generalized Binomial Arbitration Scheme

This scheme uses m busses and arbitrates in at most t stages, for $t \leq m$. With the m and t parameters determined, this scheme can arbitrate among at most $n = \sum_{i=0}^t \binom{m}{i}$ modules. The acyclic arbitration protocol and the interpretation function of this scheme are identical to those of the binary arbitration scheme of Section 3, and thus the same hardware can be used. The only difference is that generalized binomial codes from $G(m, t)$ are used as arbitration priorities.

Formally, we define this scheme $\text{GENERALIZED-BINOMIAL}(n, m, t) = \langle P, F, \text{WIN} \rangle$, for $n = \sum_{i=0}^t \binom{m}{i}$, as follows.

- $P = G(m, t)$.
- $F = \langle f_{m-1}, \dots, f_1, f_0 \rangle$, where

$$f_j(p, v_{m-1}, \dots, v_{j+1}) = \begin{cases} 0 & \text{if } \bigvee_{l=j+1}^{m-1} (p^{(l)} = 0 \wedge v_l = 1) \\ p^{(j)} & \text{otherwise,} \end{cases}$$

for $j = 0, 1, \dots, m-1$.

- $\text{WIN}(\alpha) = \alpha$, for $\alpha \in \{0, 1\}^m$.

The idea behind generalized binomial arbitration is that the interval-number of the highest competing arbitration priority bounds the number of arbitration stages. In binary arbitration, where all the 2^m m -bit codewords are used, arbitration processes can take as many as m stages, where at each stage one more bit of the highest competing arbitration priority is resolved. For generalized binomial arbitration, however, we select codewords that have at most t intervals of consecutive 1's or 0's. The following theorem uses data-dependent analysis to argue that any arbitration process takes at most r stages, where r is the interval-number of the highest competing arbitration priority, by showing that at each stage the arbitration process resolves at least one more interval of consecutive bits.

Theorem 4 *Consider a generalized binomial arbitration process on m busses. Let Q be the set of competing arbitration priorities, p be the highest arbitration priority in Q , and r be the interval-number of p . Then after s stages, for any $s \geq r$, bus b_j carries the logic value $p^{(j)}$, for $0 \leq j \leq m-1$.*

Proof. We prove the theorem by induction on r for arbitrary values of m . We use the notation $v_j[k]$ to denote the logic value on bus b_j at the end of stage k , for $j = 0, 1, \dots, m-1$ and $k = 0, 1, \dots$.

Base case: $r = 0$. The codeword p consists of m consecutive 0's, that is, $p^{(j)} = 0$ for $j = 0, 1, \dots, m-1$. Since p is the highest arbitration priority in Q , then any $q \in Q$ must also have $q^{(j)} = 0$ for $j = 0, 1, \dots, m-1$. By our assumption that all the m busses are initially in logic value 0, and since according to the acyclic arbitration protocol no module ever applies a 1 to any of these busses, the m busses remain in logic value 0 forever. In other words, after s stages, for any $s \geq r = 0$, we have $v_j[s] = v_j[0] = 0 = p^{(j)}$, for $j = 0, 1, \dots, m-1$, which proves the claim.

Inductive case: $r > 0$. The codeword p has m bits and interval-number r , and is thus of the form $p = 0^{m_0} 1^{m_1} 0^{m_2} 1^{m_3} \dots \delta^{m_r}$, where $\delta \in \{0, 1\}$; $m_0 \geq 0$; $m_j > 0$ for $1 \leq j \leq r$; and $\sum_{j=0}^r m_j = m$. We first concentrate on the first $r-1$ intervals of p , and define the set R of reduced codewords of length $\hat{m} = m - m_r = \sum_{j=0}^{r-1} m_j$, by ignoring the last m_r bits of the codewords of Q . It is easy to verify that \hat{p} , the reduced version of p , is the highest codeword in R , because we discarded the m_r least significant bits of codewords in Q . Furthermore, the interval-number of \hat{p} is $r-1$, since the last interval of p of the form δ^{m_r} was ignored. By applying the claim inductively with \hat{m} busses, the set of competing arbitration priorities R , and the highest arbitration priority \hat{p} of interval-number $r-1$, we find that after $r-1$ stages the most significant $\hat{m} = m - m_r$ busses stabilize to the bits of \hat{p} . That is, for any $k \geq r-1$, we have $v_j[k] = v_j[r-1] = \hat{p}^{(j)} = p^{(j)}$, for $m_r \leq j \leq m-1$. We now consider the last m_r busses, $b_{m_r-1}, \dots, b_1, b_0$. There are two cases to consider:

$\delta = 1$ The r th interval of p is an interval of m_r consecutive 1's, that is, $p^{(i)} = 1$ for $i = 0, 1, \dots, m_r-1$. After k stages, for any $k \geq r-1$, the most significant $m - m_r$ busses carry the bits of p , and therefore there is no l in the range $0 \leq l \leq m-1$, with $v_l[k] = 1$ and $p^{(l)} = 0$. As a result, the module with arbitration priority p applies all its last m_r consecutive 1's. Therefore, for any $s \geq r$ and $i = 0, 1, \dots, m_r-1$, we have $v_i[s] = v_i[r] = 1 = p^{(i)}$, since the busses implement a wired-OR in one stage.

$\delta = 0$ The r th interval of p is an interval of m_r consecutive 0's, that is, $p^{(i)} = 0$ for $i = 0, 1, \dots, m_r-1$. Since p is the highest arbitration priority in Q , then for any arbitration priority $q \in Q$, $q \neq p$, there must exist an l in the range $m_r \leq l \leq m-1$, with $p^{(l)} = 1$ and $q^{(l)} = 0$. After k stages, for any $k \geq r-1$, the most significant $m - m_r$ busses carry the bits of p , and therefore any module with arbitration priority $q \neq p$ disables at least its last m_r bits. As a result, for any $s \geq r$ and $i = 0, 1, \dots, m_r-1$, we have $v_i[s] = v_i[r] = 0 = p^{(i)}$, because the busses implement a wired-OR in one stage and no module applies a 1 to busses b_0 through b_{m_r-1} anymore.

Thus, after s stages, for $s \geq r$, the m busses carry the corresponding bits of p . ■

The following corollary shows that by taking $G(m, t)$, the generalized binomial codes of length m and diversity t , as arbitration priorities, we guarantee that any arbitration process completes in at most t stages.

Corollary 5 Consider GENERALIZED-BINOMIAL(n, m, t), the generalized binomial arbitration scheme. For any subset of arbitration priorities $Q \subset G(m, t)$, the corresponding arbitration process takes at most t stages.

Proof. Let p be the highest arbitration priority in Q . Since the interval-number of p is at most t , Theorem 4 guarantees that the arbitration process on Q , with p as the highest arbitration priority, takes no more than t stages. ■

The Generalized Binomial Arbitration Tradeoff

The generalized binomial arbitration scheme achieves a bus-time tradeoff of the form $n = \sum_{i=0}^t \binom{m}{i}$, which by Stirling's formula exhibits asymptotic behavior $m = \frac{1}{t} t n^{1/t}$. Figure 3 presents this bus-time tradeoff for a system consisting of $n = 64$ modules. The number of busses varies from $\lg n = 6$ to $n = 64$, and the arbitration time is in the range 1 to $\lg n = 6$ stages. Generalized binomial arbitration reduces to binary arbitration with $m = \lceil \lg n \rceil = 6$ busses, to binomial arbitration with $m = \lceil \lg n + 1 \rceil = 7$ busses, and to a modified version of linear arbitration (see Section 5) with $m = n = 64$ busses.

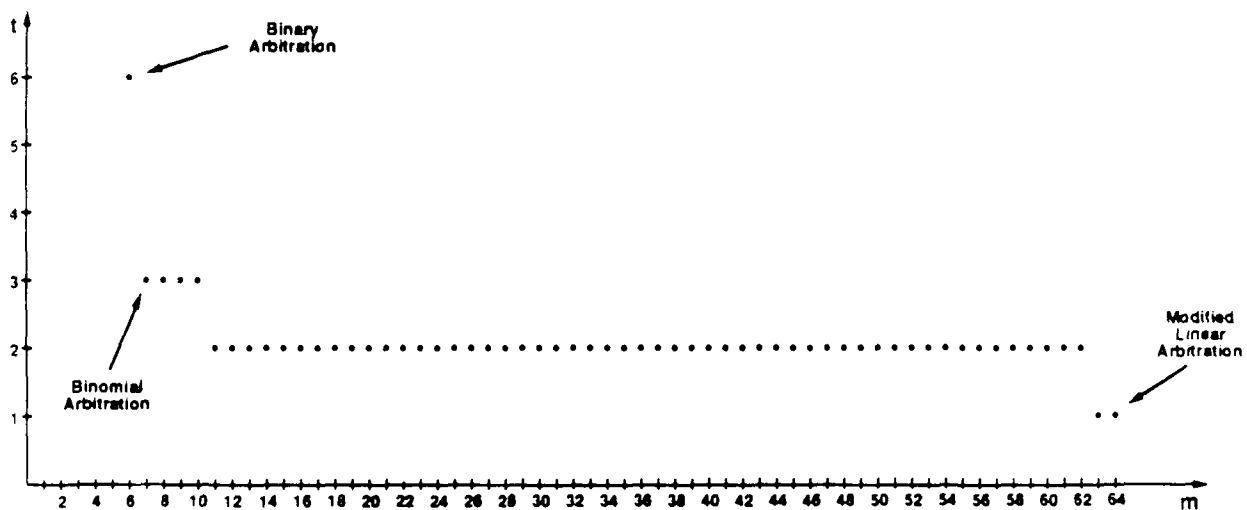


Figure 3: Bus-time tradeoff of the generalized binomial arbitration scheme for $n = 64$ modules, using $6 \leq m \leq 64$ busses and $1 \leq t \leq 6$ stages.

Figure 3 demonstrates that neither linear arbitration nor binary arbitration efficiently utilize the resources. For example, increasing the number of busses used in binary arbitration by one, results in speeding up the arbitration process by a factor of 2, as exhibited by our binomial arbitration scheme. On the other hand, allowing another time unit over linear arbitration enables reducing the number of busses from n to approximately $\sqrt{2n}$.

Notice, however, that in order to achieve another factor-of-2 improvement in the arbitration time, adding another constant number of busses to the $\lg n$ busses is not enough. Asymptotically, as n grows without bound, we need to use more than $(1 + \epsilon) \lg n$ busses, for $\epsilon > 0.232$, in order for the sum $\sum_{i=0}^t \binom{m}{i}$, with $t = \frac{1}{4} \lg n$, to be at least n . This can be verified by Stirling's formula, since when m is greater than $\lg n$ but smaller than $1.232 \lg n$, and when $t = \frac{1}{4} \lg n < m/4$, the sum of the first $m/4$ binomial coefficients $\binom{m}{i}$, for $0 \leq i \leq m/4$, does not exceed n . This demonstrates that our binomial arbitration scheme, which uses $\lg n + 1$ busses, exhibits a most economic balance, much more so than

the binary arbitration scheme. Other authors [11] have also discovered that by excluding certain codewords, the arbitration time of binary arbitration can be reduced. We, however, give the first general scheme that provides a full spectrum of bus-time tradeoff.

5 Extensions

This section contains some discussion, additional results, and directions of research concerning priority arbitration with busses.

Bus Propagation Delay, Settling Time, and Wired-OR Glitch

High-speed busses are commonly modeled as electrical transmission lines, where it takes some finite amount of time for a signal to propagate through the bus and bring the bus to a stable logic value. In addition, there are the response time of logic gates and the effect of the wired-OR glitch that need to be considered. In particular, the effect of the wired-OR glitch on bus-settling time and the use of special integration logic at module receivers to reduce this effect (see [3, 8, 16, 25]), seem to support our model.

Some authors carry out a more elaborate analysis of high speed busses (see [2, 8, 23, 24, 25]), which takes into account the distances between modules on the bus and imposes certain assumptions on the arbitration priorities. In [24, 25], for example, Taub assumes geographical ordering of module priorities and equal distances between modules on a backplane bus. Counterexamples to Taub's analysis, where these requirements are not met, have been found [2, 27]. Our model, on the other hand, is applicable to a wider classes of systems, such as data communication broadcast channels and bus systems where priorities and module locations are not predetermined and fixed.

The Asynchronous k -ary Arbitration Scheme

The linear arbitration and binary arbitration schemes of Section 3 use n -ary and binary representations, respectively, of module priorities. We can also use radix- k representation of module priorities, for other values of k , to arbitrate among $n = k^t$ modules in t units of time, using $m = tk$ busses. We sketch the asynchronous k -ary arbitration scheme here due to its simplicity and because it generalizes the linear and binary arbitration schemes rather straightforwardly. This scheme exhibits a bus-time tradeoff of the form $m = tn^{1/t}$, which is a factor of e worse than our generalized binomial arbitration scheme.

Asynchronous k -ary arbitration, for $2 \leq k \leq n$, can be described as follows. Each module is assigned a unique k -ary arbitration priority consisting of t radix- k digits. We divide the $m = tk$ busses into t disjoint groups, each consisting of k busses. During arbitration, competing module c applies the t radix- k digits of its arbitration priority p to the t groups of busses, using linear encoding of its digits on each group of k busses. As arbitration progresses, competing module c monitors the t groups of busses and disables its drivers according to the following rule: let $p^{(l)}$ be the l th radix- k digit of p and d_l be the

highest index of a bus in the l th group of busses that carries a 1. Then if $p^{(l)} < d_l$, module c disables all its digits $p^{(j)}$ for $j < l$. Disabled digits are re-enabled should the condition cease to hold. Arbitration proceeds in t stages, each of which consists of resolving the value of another radix- k digit of the highest competing k -ary arbitration priority.

Modified Linear Arbitration

A modified version of linear arbitration, which uses the same acyclic arbitration protocol of binary arbitration, achieves the same bus-time tradeoff as linear arbitration. This version is the generalized binomial arbitration scheme with $m = n$ busses and $t = 1$ time, where the arbitration priority of module c_i is $p_i = 0^{n-i-1} 1^{i+1}$, for $i = 0, 1, \dots, n-1$. This observation poses an interesting question regarding the universality of the acyclic arbitration protocol of binary arbitration.

Lower Bound for Asynchronous Priority Arbitration

The asynchronous generalized binomial arbitration scheme achieves a bus-time tradeoff of the form $n = \sum_{l=0}^t \binom{m}{l}$, where n is the number of modules, m is the number of busses, and t is the arbitration time. We conjecture that this tradeoff is optimal for our asynchronous priority arbitration model, in that no more than $n = \sum_{l=0}^t \binom{m}{l}$ modules that can be arbitrated with m busses in at most t stages.

Synchronous Priority Arbitration Schemes

In this paper we discussed the asynchronous model of priority arbitration with busses and presented several asynchronous schemes. Considering synchronous priority arbitration scheme that use clocked arbitration logic, we can show that a synchronous version of k -ary arbitration achieves a bus-time tradeoff of the form $m = n^{1/t}$ and that this tradeoff is optimal in a related synchronous model of arbitration. We can also demonstrate how to combine asynchronous combinational schemes with synchronous clocked schemes to achieve a wide spectrum of bus-time tradeoff.

Resource Tradeoffs

Resource tradeoffs of the form $m = \Theta(tn^{1/t})$, based on multiway trees and the special class of binomial trees, are discussed in [4] for a variety of problems such as parallel sorting algorithms, searching algorithms, and VLSI layouts. Asynchronous priority arbitration with busses can in fact be considered as a selection process on trees. Asynchronous k -ary arbitration corresponds to a selection process on regular trees of branching factor k , while asynchronous generalized binomial arbitration corresponds to a selection process on the more economical "modified binomial trees" of [4].

Acknowledgements

Charles Leiserson of MIT introduced me to the problem of bus arbitration, offered many meaningful insights, and assisted in the related patent application process. Steve Ward of MIT helped with issues related to backplane bus systems. Hershel Safer of MIT provided many helpful suggestions. I would also like to thank Joe Kilian, Tom Knight, Tom Leighton, James Park, and John Wolfe of MIT, Mark Manasse, James Saxe, Chuck Thacker of DEC SRC, Nicholas Pippenger of the University of British Columbia, and Marc Snir of IBM T. J. Watson Research Center, for their comments and references.

References

- [1] J. H. Anderson and M. G. Gouda, "A new explanation of the glitch phenomenon," August 1988, revised July 1989, to appear in *Acta Informatica*.
- [2] R. N. Ashcroft, R. L. Rivest, and S. A. Ward, "Counterexample to arbitration bus scheme," unpublished manuscript, MIT, September 1988.
- [3] R. V. Balakrishnan, "The proposed IEEE 896 Futurebus—a solution to the bus driving problem," *IEEE Micro*, Vol. 4, No. 4, August 1984, pp. 23-27.
- [4] J. L. Bentley and D. J. Brown, "A general class of resource tradeoffs," *Journal of Computer and System Sciences*, Vol. 25, No. 2, October 1982, pp. 214-238.
- [5] D. P. Bertsekas and R. G. Gallager, *Data Networks*, Prentice Hall, Englewood Cliffs, New Jersey, 1987.
- [6] P. L. Borill, "Microprocessor bus structures and standards," *IEEE Micro*, Vol. 1, No. 1, February 1981, pp. 84-95.
- [7] P. L. Borill, "IEEE P896—the Futurebus project," *Microprocessors and Microsystems*, Vol. 6, No. 9, November 1982, pp. 489-495.
- [8] P. L. Borill and J. Theus, "An advanced communication protocol for the proposed IEEE 896 Futurebus," *IEEE Micro*, Vol. 4, No. 4, August 1984, pp. 42-56.
- [9] D. Del Corso, "Experiences in designing the M3 backplane bus standard," *Microprocessors and Microsystems*, Vol. 10, No. 2, March 1986, pp. 101-107.
- [10] D. Del Corso, H. Kirrman, and J. D. Nicoud, *Microcomputer Buses and Links*, Chapter 5, Academic Press, London, 1986.
- [11] D. Del Corso and L. Verrua, "Contention delay in distributed priority networks," *Microprocessing and Microprogramming*, Vol. 13, No. 1, January 1984, pp. 21-29.
- [12] *The Synchronous Backplane Interconnect, Vax 11/780 Architecture Handbook*, Digital Equipment Corporation, Maynard, 1978.

- [13] M. Garetz, "P696/S100— a bus which supports a wide range of 8- and 16-bit processors," *Microprocessors and Microsystems*, Vol. 6, No. 9, November 1982, pp. 466-470.
- [14] D. B. Gustavson, "Computer busses—a tutorial," *IEEE Micro*, Vol. 4, No. 4, August 1984, pp. 7-22.
- [15] D. B. Gustavson, "Introduction to the Fastbus," *Microprocessors and Microsystems*, Vol. 10, No. 2, March 1986, pp. 77-85.
- [16] D. B. Gustavson and J. Theus, "Wire-OR logic transmission lines," *IEEE Micro*, Vol. 3, No. 3, June 1983, pp. 51-55.
- [17] J. Juang and B. W. Wah, "A multiaccess bus arbitration scheme for VLSI-densed distributed systems," *AFIPS Conference Proceedings*, Vol. 53, 1984 National Computer Conference, pp. 13-22.
- [18] J. V. Levy, "Busses, the skeleton of computer structures," in *Computer Engineering: DEC View of Hardware Systems Design*, by C. G. Bell, J. C. Mudge, and J. E. McNamara, Digital Press, Bedford, Massachusetts, 1978.
- [19] E. C. Luczak, "Global bus computer communication techniques," *Proceedings of the Computer Networking Symposium*, IEEE, December 1978.
- [20] R. Palais and L. Lamport, "On the glitch phenomenon," Technical Report CA-7611-0811, Massachusetts Computer Associates, Wakefield, Massachusetts, November 1976.
- [21] C. L. Seitz, "System timing," in *Introduction To VLSI Systems*, by C. A. Mead and L. A. Conway, Chapter 7, Addison-Wesley, Reading, Massachusetts, 1980.
- [22] A. S. Tanenbaum, *Computer Networks*, Prentice Hall, Englewood Cliffs, New Jersey, 1981.
- [23] D. M. Taub, "Contention resolving circuits for computer interrupt systems," *Proceedings of the IEE*, Vol. 123, No. 9, September 1976, pp. 845-850.
- [24] D. M. Taub, "Worst-case arbitration time in S-100-type computer bus systems," *Electronics Letters*, Vol. 18, No. 19, September 1982, pp. 833-835.
- [25] D. M. Taub, "Arbitration and control acquisition in the proposed IEEE 896 Futurebus," *IEEE Micro*, Vol. 4, No. 4, August 1984, pp. 28-41.
- [26] K. J. Thurber, E. D. Jensen, L. A. Jack, L. L. Kinney, P. C. Patton, and L. C. Anderson, "A systematic approach to the design of digital bussing structures," *AFIPS Conference Proceedings*, Vol. 41, Part II, 1972 Fall Joint Computer Conference, pp. 719-740.
- [27] S. A. Ward, private communication, May 1989.
- [28] S. A. Ward and C. J. Terman, "An approach to personal computing," *Proceeding of COMPCON*, IEEE, February 1980, pp. 460-465.

Technologies for Low Latency Interconnection Switches

Thomas F. Knight, Jr.
M.I.T. Artificial Intelligence Laboratory

Abstract

This paper presents an engineering design for a low latency high bandwidth interconnection network which will form the switching substrate for a multi-model parallel processing system. The performance is enhanced with a variety of approaches covering interconnection protocols, routing, fault tolerance, advanced packaging, and electrical interconnection techniques. The synergistic application of these technologies leads to a high performance design.

Motivation

A key performance factor in large scale parallel computer systems is the latency in processor communications. [Dertouzos 88] considers a program with available parallelism p , running on a multiprocessor of size n , with a communications latency l , measured in terms of instruction execution times. He establishes that there is a speedup linear in n if $nl \ll p$, but that this speedup approaches an asymptotic bound of p/l when $nl \gg p$.

Our parallel programming model and algorithm design can influence the available parallelism, or the average length of independently scheduled instruction sequences[1], but the latency of the communication network remains one of the fundamental characteristics of the hardware architecture.

In message passing models, the interprocessor communication latency appears as a delay in receiving messages from

remote processors [Dally 88]. In shared memory systems [Butterfly 87; Pfister 85], the latency of the communication network affects the average memory reference time. Even the addition of shared memory caches [SCI, Agarwal 88; Bistari 88] to large scale parallel shared memory systems simply moves this latency from occurring once every memory cycle to once every cache miss time. Even in SIMD architectures such as the connection machine [Hillis 85], the long latency for communications is a significant bottleneck, resulting in programmers avoiding its use when possible.

Several recent architectures supporting particular programming styles drastically lower the latency of communications to achieve higher performance. The Ametek hypercube architecture [Ametek 86], for example, achieves microsecond latencies for interprocessor communication as compared to the hundreds of microseconds for first generation hypercube processors such as the original Caltech design [Seitz 85]. Similarly, the Masspar architecture dramatically reduces the latency for large scale SIMD communications [Grondalski 87] compared to the connection machine.

Alewife

At MIT, Anant Agarwal and I are designing an architecture called *Alewife* which has as an explicit goal the support of a wide variety of parallel programming models. As such, it provides hardware support for a variety of programming styles, including several types of shared memory, message passing, and data level parallelism. To achieve this broad

[1] In the presence of code blocks of length q which can be executed independently without interprocessor communication, Dertouzos shows that the relationship is modified by substituting pq for the available parallelism p , making the speedup less dependent on the latency.

range of model support requires an extremely low latency communications mechanism. We are using this detailed design as a test-bed for the broader problem of designing extremely large, scalable parallel machines which are flexible in their programming style.

The Alewife design consists of three major components. The first component is a simple processor characterized by fast context switching, fast message dispatching, and support for data typing. The second is a cache and interprocessor communications controller capable of supporting coherent memory access in the absence of a single shared bus. Finally, the design relies on a fast, efficient communication network, called *Transit*.

The modularity of this design provides an opportunity to reuse portions of the machine as a substrate for other architectures. In particular, we are carefully defining the interface between each of the components of the architecture to allow one portion to be replaced by different or higher performance equivalents. Transit supports a carefully defined interface to the cache controller, and the cache controller presents both a uniform shared memory model and an explicit processor to processor communication model to the processors.

Transit Target Specifications

The Transit network provides uniform communications between 256 processor/memory clusters. Latency for a remote memory reference is 280 nanoseconds, and peak bandwidth is 100 megabytes/second/port. The remainder of this paper concerns the technology with which this network is constructed, and the impact these techniques have on lowering the latency of communications. We will briefly consider more advanced interconnection techniques and address the issue of scaling the design to larger numbers of processors. Because of space limitations, most of the discussion will consist of a description of the techniques Transit uses to achieve high performance, with little discussion of alternative possible designs. In many cases viable alternatives exist, but the space of possible designs is so large that it is impractical in a short paper to discuss alternatives for every decision.

Communication Protocols

Transit uses a connection based source-responsible routing protocol. The sending controller transmits a routing header and optional data forward into the Transit network, while retaining a copy of the message. The network makes a best effort to establish a communication link between the source and destination port. After transmitting the forward

message, the communications path which has been established is electrically reversed, an acknowledgment and optional data flows from the recipient to the sender. If for any reason, the attempted communications fail, then it is the responsibility of the sender to retry the connection.

The ability of the sender to retry failed communications leads to important simplifications in the routing element used in the communication switch, since we need not buffer or flow control the messages being sent to an element. Instead, the element, if it congested, is free to discard awkwardly timed messages. Similarly, failures of routing elements or the wiring between them can be handled by simply detecting the failed communications attempt using checksumming techniques, and discarding damaged data. The total failure of routing elements or interconnect is handled by redundant, randomized routing, described below. Explicit acknowledgment might seem to slow the network, but is required eventually even in networks which accept responsibility for delivering messages. Here, the reply data from a memory request, for example, can be combined with the acknowledgment.

Each port of the Transit network consists of a nine bit wide path, synchronously clocked every 10 nanoseconds. One bit is a framing bit, used to distinguish control bytes from data bytes, and the remaining eight bits are used to transmit one byte of routing information, or data. Figure 1 shows the details of the inter-chip timing of a simple transfer.

In the idle state, the sender transmits a zero framing bit each clock cycle. At the start of a message, one byte of routing data and a framing bit of one are sent into the input port. Each clock cycle thereafter, a data byte is transmitted into the input port. This forward stream of bytes is pipelined through each stage of the interconnection network, and eventually reaches the destination.

When all of the sender data has been transmitted, a distinguished byte, the *turn* byte, (all one's with a zero framing bit) is transmitted. This is a signal to reverse the data flow in the network. On receipt of the turn byte, each stage of the network starts pipelining data back to the original sender. When the turn byte reaches the destination, a complete reverse path has been set up allowing data to flow from destination to the sender. The destination transmits an acknowledgment, followed by any number of data bytes. The framing bit in the reverse direction is used to signal the completion of data transfer.

Status information is available to the sender as a side effect of this sequence. Because of pipelining, a D stage switch has 2D clock periods of delay following the sender's transmission of the turn byte and prior to the arrival of the acknowledgment. During this period, each stage of the

Unblocked Data Transfer Timing

- (1) Bytes driven toward the input port
- (2) Bytes driven by the input port
- (3) Bytes driven by the output port
- (4) Bytes driven toward the output port

R Routing Byte
F Forward Data Byte
T Turn Byte
S Status Byte
C Checksum Byte
B Backward Data Byte

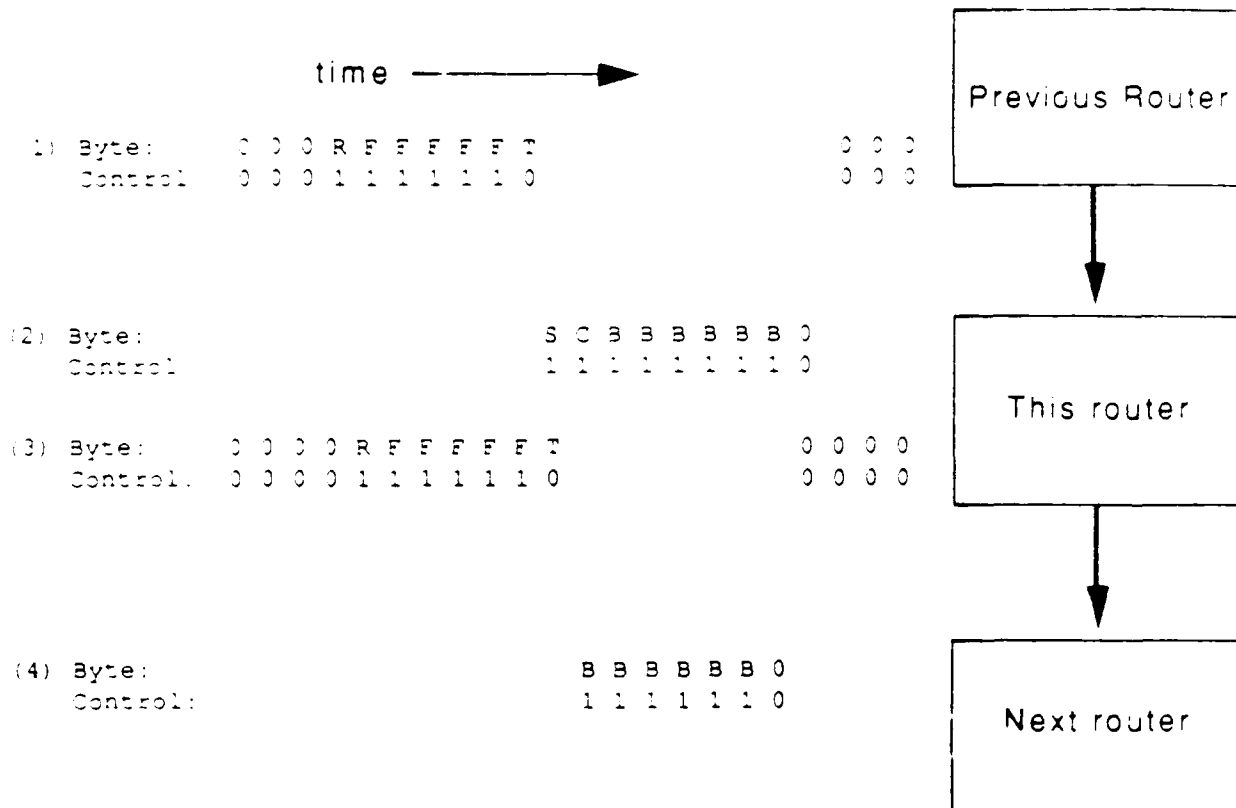


Figure 1

interconnection switch transmits a pair of status bytes back to the sender, indicating which (if any) of the output ports were assigned to the connection leaving this switch stage, and a checksum for the message at this stage of the switch. The status information is used by the sender to determine the exact path through the switch this message was routed with, to determine where a message was garbled in transit, and to determine at which switch stage a message was discarded, if it was thrown away.

Interconnection Topology

The Transit network consists of a four stage, radix four omega network, providing 256 possible destinations. Each routing element is an eight input port, eight output port switching component. The eight input ports are

interchangeable, and the eight output ports are paired in four groups of two ports each. An input message is routed to one of the two available output ports in the direction specified by two bits of the routing byte. Once this routing is performed, the path which is set up will remain assigned until the connection is dropped. If neither of the two output ports in the desired output direction is available, the message is discarded.

The wiring of the port from each stage of the switch to the next is arranged so that the data wires are rotated by two bits. This permutation of the data wires allows the two bit field of the routing byte seen by each of the four stages of the switch to differ, routing the message on all eight bits.

The pairing up of output ports in the routing element

provides an important fault tolerance feature of the design. If both output ports in a given direction are available when a message is to be routed, a pseudo-random number generator is used to arbitrarily choose between them. This assures that the path taken through the switch on an attempt to retransmit a message after failure will, with high probability, take a different path than the first try. This path redundancy allows fault tolerance to be built into the network at very low overhead. Ideally, the two output ports which go in logically identical directions should be wired to physically distinct routing elements to provide better fault coverage. This is possible in all but the final stage of the switch, where all messages destined for a particular processor must flow through one routing element. The necessity to wire this final stage differently is in conflict with the desire to wire all stages with the same permutation, for reasons which we describe below in the section on packaging.

The choice of four pairs of output ports as a routing element design also has important implications for the statistical success of the routing process. This issue is discussed in detail in the section below on performance.

Packaging Issues

The packaging of high performance systems has an extreme impact on their speed -- to the extent that system level design is often dictated by available packaging technology. The Transit network is packaged using a unique three-dimensional wiring technology which allows roughly equivalent wiring density in all three dimensions. The approach consists of using conventional printed circuit boards, with a 50 ohm controlled impedance stripline structure, for two of the three dimensions. For the third dimension of wiring, these boards are layered on top of one another, as shown in figure 2.

Contact between the boards is provided with *button boards* (Smolley 85), a term describing compliant fine wire fuzz buttons pushed into blank, drilled printed circuit board material (figure 3). These buttons, formed by compressing 25 micron wire into a cylindrical die 20 mils in diameter by 40 mils high, are used on staggered, 50 mil centers, to provide extremely dense connectors between layers of the packaging. Because of the short distances involved, impedance mismatch is minimal if care is taken with ground wire density.

Components are packaged into this structure by mounting them on carriers also fabricated from standard PC board materials. A recessed cavity is used to hold the die, which is then wire bonded or tab interconnected to the carrier. The carrier is unlike normal integrated circuit packages in that its pins are simply flat pads located on both the top and

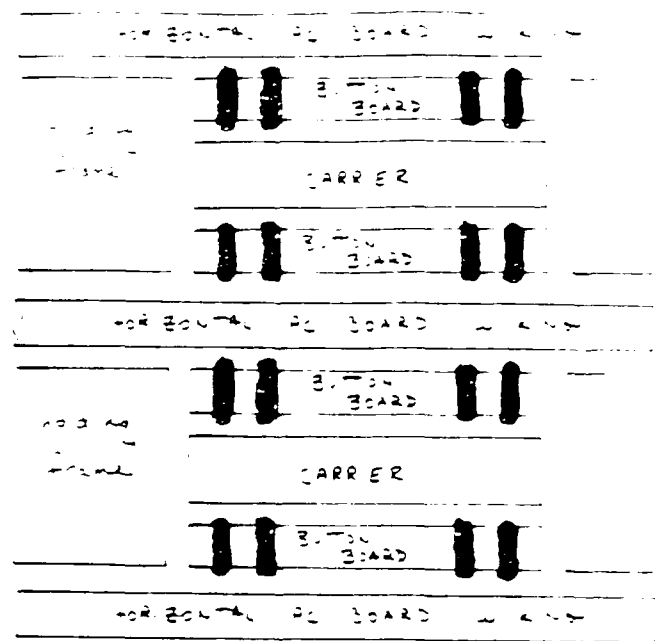


Figure 2

bottom of the carrier board. Thus terminals of the die are accessible from top or below, and wires, if necessary, can be simply routed through the carrier with no connection to the die. The carrier board provides a controlled impedance environment for signals up until the bond to the die. In addition, the carrier provides low inductance power and ground plane decoupling capacitance through integral layer proximity, as well as locations for mounting explicit ceramic bypass capacitors. Through holes are provided in the carrier for vertical fluid cooling channels.

Component carriers, together with upper and lower button



Figure 3

board connectors, are placed into a holding frame which provides two dimensions of horizontal alignment. The holding frame, with its button boards and carriers, forms a layer in the stack. Stack-wide printed circuit boards typically alternate with layers of holding frames and chips, providing a compact, dense, three dimensional means for building relatively small (30x30x30 cm) three dimensional structures.

The logical structure of the four stage radix four omega network is mapped onto the three-dimensional package by packaging each stage of the routing network in a separate layer of the stack. Signal flow through the network is thus logically in the vertical direction, from one layer to the next. The omega topology has the valuable property of having identical wiring patterns between stages of the network; this property is exploited in the stack by replicating the interconnection structure of each stage multiple times. Ideally, then, the stack consists of a structure alternating a fixed omega wiring permutation of signals in the horizontal direction, with layers of routing elements. Four such wiring routing element pairs complete the three dimensional stack. Figure 4 shows the wiring pattern for the horizontal wires in one of the layers of the stack. Each line represents a pair of ports; this figure shows a the wiring for a 64 port network.

The vertical signal flow means that inputs to the switch structure are available at the top, and that outputs are available at the bottom. Because we wish to use this network as a processor to processor communication switch, the inputs and outputs must be available in physical proximity. This is solved by routing the network outputs back through the stack vertically on additional wiring channels. These channels take up little space, since there is no horizontal wiring associated with them.

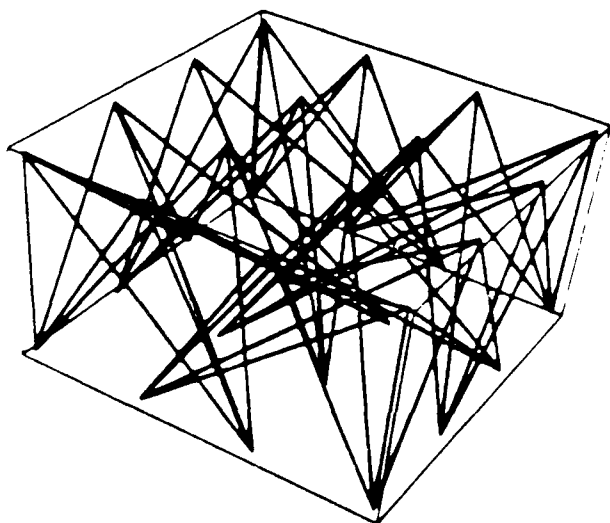


Figure 4

Providing electrical power to the circuits and removing waste heat remain significant issues. The fuzz buttons are excellent, low resistance connectors, and, because of the large number required between boards for impedance control, exist in abundance to provide a low inductance path vertically between boards. Horizontally, power is provided using integral power-ground plane structures within the controlled impedance boards. These planes also provide important low inductance power supply filtering. Power is brought into the stack with power lugs mounted on boards at the center (vertically) of the stack which extend horizontally beyond the normal boundary of the stack.

Heat is removed from the stack using FC-77 Fluorinert liquid flowing vertically through the stack. The entire stack is normally run immersed in Fluorinert, and pressurized fluid is pumped into a distribution manifold at the top of the stack. This manifold also acts as one of the pressure plates which apply compressive force to mate the large number of button board contacts. The high heat capacity per unit volume of liquid cooling relative to air cooling dictated its use in the high density structure. Modest flow rates (2 gal/min) should be adequate to cool our prototype system.

As a result of the aggressive packaging used in this design, the longest wires are approximately 45 centimeters. Modest cost, easily fabricated, low dielectric constant PC board materials such as Norplex cyanate esters, have a dielectric constant of 3.1. The wire delay of the longest paths in the design is thus approximately 2.65 nanoseconds.

This composite structure has many advantages over conventional packages. First, since it is three dimensional, the wire length for a given wiring density is substantially smaller than structures otherwise achievable using two dimensional packaging, backplanes, and cables. Second, it is easily repairable by disassembly of the stack, since it involves no soldering or other permanent connections. Third, though it might seem awkward to debug, simple boards can be constructed which, when added to the stack between particular layers, allow signals in that layer to be examined.

Electrical Issues

An early decision was to totally abandon the idea of using multi-drop bus like electrical structures in the design. The drastic reduction in signal speed and line impedance due to capacitive loading of the transmission lines in even carefully engineered systems argued strongly that point to point communications be used.

A dominant electrical design issue was how to drive the very large number (23,000) of terminated signal wires in the switch. A 50 ohm impedance level is dictated by practical wire geometries, and could not, in any case, be raised by more than a factor of two. With standard CMOS signal swings of five volts, the parallel termination of a single wire would dissipate a half watt! We reduce this power dissipation by a factor of 50 by lowering the signal swing to one volt, and by series terminating the transmission lines. The series termination allows the impedance seen by the output driver to be twice the impedance of the line, but is applicable only to point-to-point wiring.

The series termination resistance is provided within the pullup and pulldown transistors of the output driver, as described in [Knight 88]. Our current design differs a little from the technique described in that paper in that it uses a digitally controlled D/A like structure to vary the output transistor resistance. The use of resistive pullup and pulldown devices has important speed implications, since the devices need not (must not) be large devices, and hence can be driven far more quickly than conventional low impedance output driver transistors. Providing the terminating resistors on-chip also has the large advantage of eliminating 23,000 discrete resistors from the stack, and allows for electrical compensation of both the driver impedance and the line impedance against manufacturing variation.

The one volt logic swing of the output driver is compatible, in magnitude, with the approximately one volt swing of ECL logic families. As a result, the use of small quantities of small scale ECL logic for applications such as clock buffers and I/O interfacing is practical, using a pair of offset power supplies for the ECL circuitry.

One of the difficulties we have encountered is the very low efficiency of one volt power supplies. At these voltages, the voltage drop of a silicon diode (.7 volts) becomes a major source of power supply inefficiency. Synchronously switched MOS power devices used as rectifiers will solve this problem, but there is as yet no commercial demand for this development. As VLSI devices scale to smaller dimensions, the need for high efficiency, low voltage power supplies will become very evident.

We are currently investigating two techniques for clock distribution. The conventional approach is to use multi-stage clock fanout with equal length and matched delay transmission lines to each network element for delivery of a time aligned clock signal. A second approach of treating the clock signal as a single node, wired in a highly interconnected three-dimensional grid may offer some

advantage. The grid must be driven at multiple locations (perhaps every 5-10 cm in all axes) and treated as a lumped capacitive load. Since the clock waveform is of a single frequency, we can consider the possibility of resonating this capacitance with a tuned inductance to reduce clock distribution power.

Performance

One of the advantages of the unbuffered style of communication network is ease of performance analysis. Since the network timing is determined entirely by the pipeline delay, the latency for successful messages is easy to calculate. Since the system is memoryless except at the sender, the probability of routing success within the network can be calculated quite easily, using the analytic techniques described in [Knight 89].

A typical message might consist of a remote memory read access. Such a request would send an address forward through the Transit network, cycle the remote memory, and return an acknowledgment and the read data. For 32 bit address and data, the forward message is five bytes long, and the reverse message is five bytes long. A two byte checksum will likely be added to these message lengths, although these are indistinguishable from data to the network. The pipeline delay of the network is four clocks, so the remote access is complete in $7+4+7+4 = 22$ cycles. By making optimistic assumptions about the success of checksumming the data, we can overlap a portion of the forward message delivery with the cycling of the remote memory system. As soon as two bytes of address are received, we can initiate a RAS cycle on the remote memory system, and start the memory cycle in parallel with receipt of the remainder of the address. Similarly, the acknowledgment byte may be sent prior to having access to the read data. This gives 60 nanoseconds at the remote processor/memory pair to perform a memory RAS/CAS cycle and obtain the data.

The probability of successfully routing through the Transit network as a function of input loading is shown in figures 5 and 6. The input loading is the probability that an input port has a message being sent or received. The best that can be achieved without combining approaches is the non-blocking behavior of the crossbar. Figure 5 shows the performance of a crossbar network with one output port to each logical destination. For comparison, the eight stage omega network constructed out of 2 by 2 switch elements, and the four stage omega network constructed out of 4 x 4 switch elements are also shown. The Transit network, further limited to a single output port per logical destination is shown on this same graph. The extra output ports between switching elements leads to behavior very close to the ideal behavior of the crossbar.

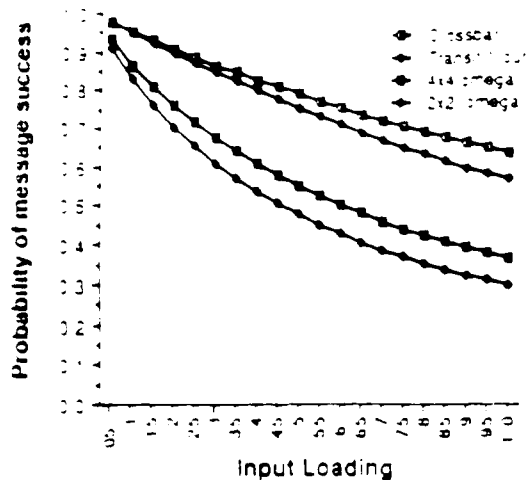


Figure 5

Similarly, in figure 6, we show the ideal behavior of a large non-blocking $N \times 2N$ crossbar which allows two output ports per logical destination. Below it we show the performance of the Transit network, again demonstrating performance close to the behavior of a crossbar.

The reason for this good performance lies in the choice of network element -- particularly in the availability of multiple output paths travelling in a single logical direction. The performance of the network from a probabilistic standpoint could be improved yet more by constructing a switching element with eight inputs and two clusters of four outputs each, where each of the four ports in a cluster travelled in a logically equivalent direction. The disadvantage of this approach is the doubling of the number of stages in the network, since only one bit worth of routing is performed per stage of the network. The choice of the element for Transit was dictated by a desire to minimize the pipeline delay of the network while maintaining good probabilistic performance.

Technology Extrapolation and Limits

The approach of constructing large multi-stage omega networks becomes infeasible at a point not much larger than the network we are constructing, due to the exponential growth of wiring. For processor networks larger than can be packaged with short wiring, the architect (and ultimately the programmer) must face the importance of locality in constructing very large parallel machines. Perhaps the most elegant approach to acknowledging the necessity for this locality is the *fat-tree* [Leiserson 85] approach. A fat-tree can be thought of as a multi-stage omega network where local transactions are successively isolated from more global transactions. The more global transactions are routed, through successively more narrow

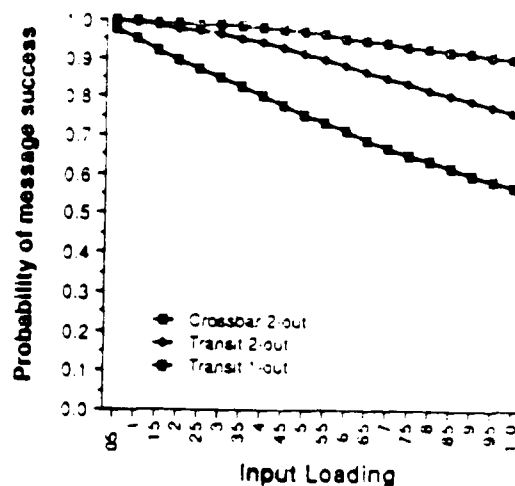


Figure 6

channels, towards a global switching array. Finally, they arrive at the most global (root) node of the network, and, from there, may be delivered to any location. The narrowing of the channels as the root is approached allows this network to scale to very large arrays, at the cost of latency, and of limited ability to communicate globally.

We can construct fat-tree based routing networks from the stack structure described above for Transit by adding one additional routing stage per stack. The purpose of this routing stage is to isolate messages destined for more global stages of the switch from those that may be delivered locally. The more global messages are routed to the bottom of the stack, where they connect to a set of flexible printed circuit board layers used as cabling between stacks. The other end of these flexible PC board cables is routed to the top of another stack, along with the global signals from three additional stacks. Outputs of the global stack similarly are channeled back to the local stacks. This approach of constructing a fat-tree like structure from a tree of high performance routing stacks appears to be an effective way of building networks which combine high performance, an ability to take advantage of locality, and scalability to tens of thousands of high performance processors.

Two alternative electrical techniques for communicating between routing elements appear to be important alternatives. One is the approach of Rettberg, Glasser and Basset [Rettberg 87] for eliminating the reliance on low clock skew in the signal paths. Future versions of the Transit network will likely require an approach similar to this, especially if the wiring between stacks is long enough to impose delays large compared to the anticipated clock rate.

Another approach which we are devoting some attention to

is the notion of transmitting data between chips by use of modulated microwave carriers. The advantage of this scheme is the elimination of the DC component of the digital signal, transforming a broadband digital signal into a narrowband RF signal. For the same reasons that modems are an appropriate technique for transmitting data on long distance telephone lines, the use of narrowband data transmission allows many electrical tricks which are otherwise not available. Transformers, power splitters, limiters, stub tuning of transmission lines, and automatic gain controls can all be used to good advantage in communicating these signals. The high dispersion of transmission lines associated with the series resistance of the line is a much smaller problem when the range of frequencies is less than an octave. Finally, and perhaps most compelling, the connection of signals from one physical structure to another need not be done with wires, but may be done with the intrinsic capacitance of adjacent metal contacts. A chip, for example, might not need bond wires for the signals, but only for power and ground distribution.

Summary

We have presented an initial engineering design for a high performance processor to processor interconnection switch intended as the substrate for a programming model independent computer architecture. Some of the key elements of this approach have already been tested in prototype form, and we are actively pursuing a complete prototype.

Acknowledgments

This research is supported in part under DARPA contract number N00014-87-K-0825 and in part by the Vinton Hays Endowment. The author is supported by an ITT Career Development Professorship. The author would like to acknowledge assistance from Anant Agarwal, Henry Minsky and Andre Dehon, and valuable discussions with Charles Leiserson, Andrew Berlin, Michael Dertouzos, Lance Glasser, William Dally, Alex Krymm, and Robert Marker.

- | | |
|---------------|--|
| | nical report CMU-CS-88-204, December 1988. |
| Butterfly 87 | BBN, Butterfly Products Overview, Technical Report, BBN Advanced Computers, Inc., October, 1987. |
| Dally 88 | Dally, William J., Fine Grain Message Passing Concurrent Computers, 1988 |
| Dertouzos 88 | Dertouzos, M.L., A Fundamental Relation on Multiprocessor Performance, |
| Grondalski 87 | Grondalski, R., A Chip Set for a Massively Parallel Architecture, IEEE International Solid State Circuits Conference, February 1987, pp. 198-199. |
| Hillis 85 | Hillis, D., The Connection Machine, Cambridge, Mass., MIT Press, 1985. |
| Knight 88 | Knight, T.F., and Krymm, A., A Self Terminating Low-Voltage Swing CMOS Output Driver, IEEE J. of Solid State Circuits, Vol. 23 No. 2, April 1988. |
| Knight 89 | Knight, T.F., Routing Statistics for Unqueued Banyan Networks, M.I.T. A.I. Lab Memo 1102, June 1989. |
| Leiserson 85 | Leiserson, Charles E., Fat Trees: Universal Networks for Hardware Efficient Supercomputing, IEEE Tr. on Computers, Vol. C-34 No. 10, October 1985 |
| Pfister 85 | Pfister, G., et al., The IBM Research Parallel Processor Prototype (RP-3): Introduction and Architecture, 1985 Int'l Conference on Parallel Processing, August, 1985, pp. 764-771, see also ff. |
| Rettberg 87 | Rettberg, R., and Glasser, L., Digital Phase Adjustment, U.S. Patent No. 4,700,347, October 1987. |
| SCI | Scalable Coherent Interface is a study group for a future IEEE interconnection standard (P1586). See Krisiiansen, E.H., et al., Scalable Coherent Interface, Eurobus Conference Proceedings, Munich, May 1989. |
| Seitz 85 | Seitz, C.L., The Cosmic Cube, Communications of the ACM, Vol. 28 No. 1, January 1985. |
| Smolley 85 | Smolley, R., Button Board, A New Technology Interconnect for 2 and 3 Dimensional Packaging, International Society for Hybrid Microelectronics Conference, November 1985. |
| Agarwal 88 | Agarwal, A., et al., An Evaluation of Directory Schemes for Cache Coherence, 15th International Symposium on Computer Architecture, June 1988. |
| Ametek 86 | Ametek System 14 User's Guide, C Edition, Ametek Computer Research Division, Arcadia Ca., 1986. |
| Bisiani 88 | Bisiani, R., et al., Coherent Shared Memory on a Message Passing Machine, CMU tech- |